VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝKONNOSTNÍ A REGRESNÍ TESTOVÁNÍ NÁSTROJE ZONER PHOTO STUDIO X

PERFORMANCE AND REGRESSION TESTING OF ZONER PHOTO STUDIO X

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATĚJ KONOPÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

143928

Ústav: Ústav inteligentních systémů (UITS)
Student: **Konopík Matěj**
Program: Informační technologie
Specializace: Informační technologie
Název: **Výkonnostní a regresní testování nástroje Zoner Photo Studio X**
Kategorie: Analýza a testování softwaru
Akademický rok: 2022/23

Zadání:

1. Nastudujte výkonnostní testování, automatizaci testů a regresní testování. Seznamte se s produktem Zoner Photo Studio X (ZPS X) a jeho funkčními nebo výkonnostními problémy.
2. Navrhněte proces kontroly dílčích části produktu ZPS X založeném na regresním testování. Navrhněte nutné úpravy zdrojových kódů a překladového systému pro automatickou kontrolu zdrojových kódů.
3. Navrhněte a implementujte automatické testy demonstrující použitelnost navrženého procesu kontroly kvality. U výběru automatických testů uvažujte testování výkonnosti.
4. Ověřte základní funkcionalitu odhalení výkonnostních či funkčních problémů založeném na reálném případu.

Literatura:

- Ammann, P., Offutt, J. *Introduction to Software Testing*. Cambridge University Press, 2008, 322 s. ISBN 978-0-511-39330-3.
- ISO/IEC/IEEE 29119-4:2015(E) Software and system engineering -- Software testing -- Test techniques.
- Myers, G. J., Sandler, C., Badgett, T. *The Art of Software Testing. 3rd Edition*. Wiley. 2011. ISBN 978-1118031964.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 3.11.2022

Abstrakt

Tato práce se zabývá problematikou testování operací nad soubory ve formátu RAW z moderních digitálních fotoapárátů v rámci produktu Zoner Photo Studia X. Využívá se automatizovaných regresních testů s grey box přístupem pro kontinuální kontrolu kvality výstupních obrazových dat. Byl také vytvořen unifikovaný systém pro testování modulu Zoner Photo Studia X zodpovídajícího za převody RAW souborů. Systém představuje otevřevené rozhraní pro nové testy obrazových dat, automatické spouštění a zabudovanou analýzu i reporting. Díky tomu je snadnější vytvářet nové testy a zvyšovat tak výslednou kvalitu produktu.

Abstract

This thesis deals with the problematics of testing the manipulation of RAW format files from the modern digital cameras in Zoner Photo Studio X. Automatic regression tests are used together with the grey box method to continually control the quality of the output image date. A new testing system was created that is used to test the RAW conversion module. This system brings and open interface to easily and quickly integrate new image data tests, It also features automatic launching, analysis and reporting of the results. Thanks to this new system, the implementation of new test will become much easier and this will moreover increase the quality of the product.

Klíčová slova

Testování softwaru, regresní testování, blackbox testování, soubor RAW, Zoner Photo Studio X, testovací systému, testovací infrastruktura, výkonnostní testování, zaručení kvality, cyklus vývoje software, digitální fotografie, Bayerův filtr, třídy ekvivalence, vazební hodnoty, agile development, digitální kamery, úpravy fotografií.

Keywords

Software testing, regression testing, blackbox testing, RAW file development, Zoner Photo Studio X, test system implementation, performance testing, quality assurance, software development lifecycle, digital photography, Bayer filter, equivalence classes, boundary value analysis, agile development, digital cameras, photo editing

Citace

KONOPÍK, Matěj. *Výkonnostní a regresní testování nástroje Zoner Photo Studio X*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Výkonnostní a regresní testování nástroje Zoner Photo Studio X

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Aleše Smrčky. Další informace mi poskytli Aleš Hasala, Dušan Doležal, Petr Minář a David Raška. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Matěj Konopík
10. května 2023

Poděkování

Rád bych poděkoval všem z firmy Zoner, kteří se na možnosti tuto práci vyhotovit podíleli. Jmenovitě Pavlovi Fridrichovi a Michalovi Prouzovi. Další díky patří vývojářům, kteří mi při práci poskytovali četné doporučení a ukazovali mi správný směr. Navíc jim patří dík za důvěru v moje řešení. také musím poděkovat Libuši Konopíkové za podporu

Obsah

Kapitola 1

Úvod

Při testování softwaru nikdy neexistuje nástroj, který by byl uplatnitelný pro všechny oblasti této disciplíny. Cílem mojí práce je zavést testování pro modul Zoner Photo Studia X a vytvořit nástroj na míru jejich velice specifickým potřebám tohoto profesionálního fotosoftwaru. Modul, jehož testování se ve své práci věnuji, zodpovídá za převody fotografií ve formátu RAW do standardních obrazových formátů a v průběhu toho se pokusit co nejlépe otestovat složitý tok dat, který se skrývá pod zdánlivě snadnou konverzí formátu souboru Formát RAW ve skutečnosti není jen obyčejný soubor, u kterého byste věděli úplně vše už z načtení jeho hlavičky. RAW je velice specifický formát, který navíc ani nemá pevný standard a neotevřete ho dvojím kliknutím z plochy počítače. V podstatě se jedná o surová data, která byla zaznamenána na CMOS senzor digitálního fotoaparátu a pak odeslána na SD kartu bez dalších velkých interakcí. Tyto soubory nachází využití hlavně v rukou profesionálních fotografů, kteří potřebují od svého foťáku více než komprimovaný JPEG. Jelikož se do RAWu souborů zapisují přímo digitalizované signály ze senzoru, jenž jsou pouze informace o osvětlení plochy, čistý RAW ani nenese barvy, kromě zabudovaného JPEG souboru, který slouží k nahlížení. Mým cílem je pomoci vývojářům Zoner Photo Studia X zajistit, že vášnivý fotograf po otevření tohoto souboru v počítači uvidí to co, opravdu vyfotil. Zpracování surových dat ze senzoru je docela náročná disciplína a pro producenty fotografického softwaru je to o to horší, jelikož výrobci digitálních kamer si svá technologická tajemství v rámci obchodní soutěže nechávají pro sebe. Existuje množství bodů, kde při zpracování RAW souborů může číhat chyba. Tato práce se zaměřuje na tvorbu automatizovaných regresních a výkonnostních testů které zaručí, že všechny podporované formáty RAWů půjdou bez problému přeložit a jejich obrazová informace bude stabilní. Mimo samotné testování se také zaměřuji na vývoj testovacího systému, který by poskytl unifikované rozhraní pro tvorbu nových testů ve společnosti, kde se příliš netestuje. Rád bych, aby v testování modulu, na který se soustředím, zavládl organizovaný řád a aby byla tvorba nového testu snadná a rychlá. Výzvou je, že se nejedná o konzervativní způsob testování, ale o proces ověřování, zda se soubor RAW po převodu na standardní bitmapový formát opravdu zobrazuje ve fotografickém softwaru tak, jak jej viděl fotograf v hledáčku. Největší výzvou při testování formou black box, či gray box, je zvládnout korektně nastrukturovat, spravovat a používat vstupní data testovaného systému. Mimo samotné testování se v této práci zabývám i analýzou vstupních dat, která jsou pro mě v tomto případě soubory RAW ve všech jejich kombinacích. Konečným cílem je návrh a implementace regresních testů pro kontrolu obrazových dat, která jsou výstupem testovaného modulu pro konverzi RAW souborů Zoner Photo Studia, na který se tato práce zaměřuje.

Kapitola 2

Současný stav vývoje produktu Zoner Photo Studio X a jeho testování

2.1 Pozadí produktu Zoner Photo Studio X

Zoner Photo Stuido X je program pro správu, úpravy a pokročilou fotomanipulaci snímků z digitálních fotoaparátů pro platformu Microsoft Windows. Tento program, respektive verze X, jenž nese číselné označení verze 19, je poslední a zároveň kontinuálně vyvíjený produkt. Jeho vývoji se věnuje celá softwarová divize firmy ZONER a.s. sídlící v Brně. Softwarová divize vznikla v roce 1993 s cílem přinést na tuzemský trh kvalitní fotosoftware v českém jazyce. [1] První produkt, ze kterého se později Zoner Photo Studio X vyvinulo, byl Zoner Media Explorer, který sloužil hlavně k procházení fotografií. V roce 2002 byl vydán Zoner Media Explorer 5, který se těšil relativně velkému úspěchu na tuzemském trhu. Ve stejné době vzniká také Zoner Callisto, vektorový grafický editor, který měl prodejní úspěch i v zahraničí pod názvem Zoner Draw. Zoner Photo Studio X, jemuž se ve svojí práci věnuji, vzniklo poprvé v roce 2004 s označením verze 7, jelikož se vyvinulo právě ze Zoner Media Exploreru 6. [2] Verze X je tedy devatenáctým a posledním vydáním tohoto produktu, jelikož se s touto verzí přešlo na systém kontinuálního vývoje. Zároveň se s touto verzí změnil i licencovací systém. Dříve si zákazník koupil software jako produkt se standardním licencováním skrze licenční klíč a tak tomu bylo až do verze 18. Verze 19, respektive X, přešla na *subscription-based model*, kdy zákazník platí za určitou dobu přístupu k produktu. U Zoner Photo Studia X je standardní doba předplatného jeden rok. Díky tomu jsou alokovány finanční zdroje pro kontinuální vývoj produktu a zákazník má zároveň dostupnou plnou podporu a záruku na funkčnost programu po dobu aktivního předplatného. S tímto licencovacím modelem je tedy potřeba zaručit kvalitu vydávaného produktu, aby zákazník platící za jeho využívání měl jistotu, že se něj může spolehnout.
Tím, že je Zoner Photo Studio vyvíjeno relativně dlouhou dobu se v bázi zdrojových kódů lze setkat s takzvanými *legacy* řešeními, tedy sekcemi kódu nebo moduly, které byly v minulosti validovány a jsou považovány za funkční, ale byly vyvinuty někdy v minulosti a vývojář zodpovědný za jejich správu například již ve společnosti nepracuje. Legacy kód tak ztrácí flexibilitu a přináší vývojářům jistý technologický dluh, který poté může ovliv-

[1] Dostupné z https://www.zoner.eu/fotograficky-software
[2] Dostupné z https://wiki.zoner.com/software:history?id=software:history

nit rychlost dalšího rozvoje, nebo dokonce ovlivnit spolehlivost přidružené funkcionality v programu. To stejné může nastat při změně v knihovnách, které používáme. Například neohlášená změna rozhraní a výstupů operací v externí knihovně, jenž se používá pro převod RAW souborů popsaném v sekci 3.5, může ovlivnit chování konverzního modulu, který je nad touto knihovnou vybudován. Ve své práci se zabývám testováním modulu pro převod snímků z digitálních fotoaparátů ve formátu RAW do takových obrazových formátů, které je možné zobrazit. V tomto modulu momentálně neexistují celkové systémové testy a můj cíl je zaručit, že převody RAW souborů ze všech podporovaných digitálních fotoaparátů budou korektní i po aktualizaci jakékoliv části RAW převodníku v produktu Zoner Photo Studio X.

2.2 Vývojový cyklus a stav testování

Jak bylo řečeno, Zoner Photo Studio X je kontinuálně vyvíjeno. Vývojový cyklus využívá agile metodik a skládá se ze dvou půlročních sprintů, jejichž produktem je větší aktualizace programu, které přináší nové funkce, jak na základě žádostí uživatelů, tak na základě interních plánů a požadavků. Dále se uprostřed sprintů mezi těmito aktualizacemi vydávají záplaty, které řeší nalezené problémy v ostře nasazené verzi produktu. Veškerá evidence úkolů i hlášení chyb je integrována do systému Asana, do nějž mají přístup všichni zaměstnanci divize i napříč odděleními. Jako verzovací systém se používá z legacy důvodů Apache Subversion (SVN). Jelikož je většina báze kódu psaná v programovacím jazyce `C++`, vývojáři používají integrovaného vývojového prostředí Microsoft Visual Studio.
Všechny produkční kanály pro vydávání aktualizací jsou detailněji popsané v sekci 2.7. Velké aktualizace procházejí interním alpha testováním, ve kterém se zapojují zaměstnanci a následným beta testováním, do kterého jsou zapojeni i vybraní uživatelé, kteří o to projeví zájem. Ve fázi beta testování zapojení uživatelé používají produkt takovým způsobem, jakým jsou zvyklí a v případě nalezení chyby dochází k zavedení záznamu do systému pro správu projektu. Následuje interní analýza chyby a její odstranění před ostrým vydáním. Tyto chyby jsou často spojené s novými funkcemi přidanými v rámci té které aktualizace a místy také s nekonzistencí uživatelského rozhraní. Tímto způsobem lze snížit počet chyb, se kterými by se typický uživatel setkal v rámci běžného užívání programu, ale vzhledem k rozsahu funkce programu se často nepodaří pokrýt v ostrém vydání všechny chyby a při nalezení takových chyb v ostrém vydání jsou třeba hotfixy. Jak již bylo zmíněno, moje práce se zaměřuje na regresní testování modulu převodů RAW souborů z podporovaných fotoaparátů.[3] Během beta testování je možné, že uživatelé odhalí problémy při převodu RAW souborů ze svého osobního fotoaparátu, ale nelze očekávat, že se takto odhalí chyba u všech podporovaných fotoaprátů. Bylo by potřeba mít bázi uživatelů se všemi fotoaparáty, což není realistické. Testovací systém, který je v rámci této práce navržen, se zaměřuje právě na tuto problematiku a na zachycení chyb, které by v tomto ohledu mohly vystát. Detailní popis problematiky převodů RAW souborů je k dispozici v kapitole 3.
Aktuálně v softwarovém oddělení firmy ZONER a. s. existují pouze unit testy, které si vývojaři sami implementují a spouští nad vlastními subprogramy a knihovnami pro Zoner Photo Studio X. Využívá se při tom C++ knihovny Boost a integrace do IDE Visual Studio.

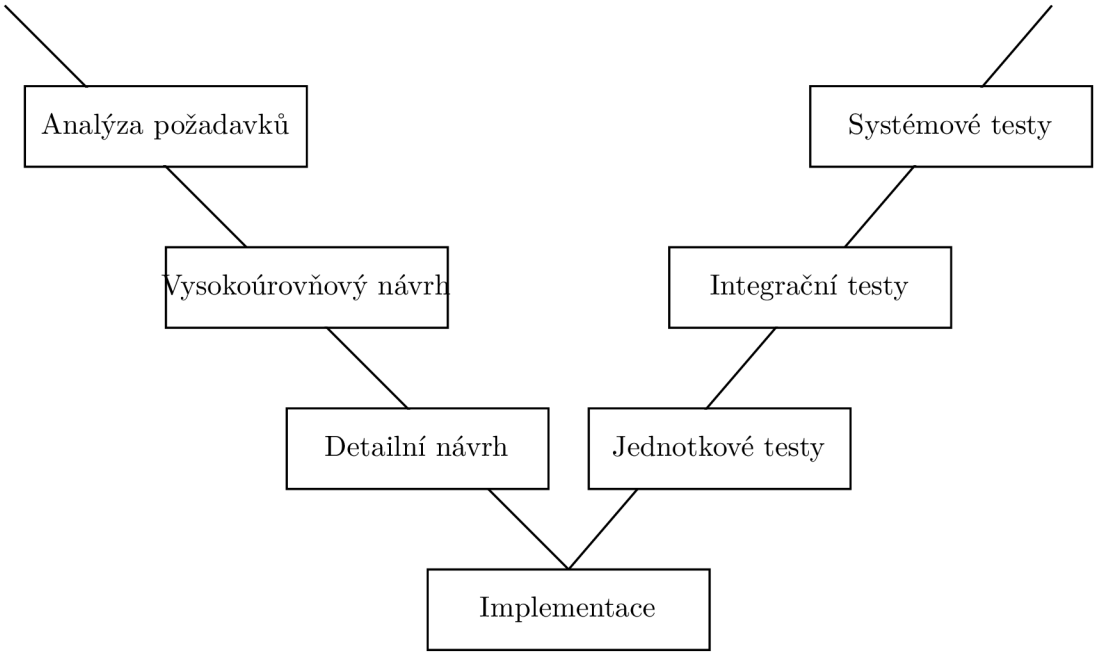
[3] Seznam je k nalezení na stránkách https://www.zoner.cz/podpora/podporovana-technika.

2.3 Zavedení testování pro Zoner Photo Studio X

Hlavním cílem této práce je navrhnout testovací systém, který pro produkt Zoner Photo Studio X přinese integraci existujících testů, možnost implementace nových testů a navíc návrh a implementaci integrovaných regresních a výkonnostních testů nad komponentou pro převody RAW souborů3. Navržený testovací systém bude tedy schopen spouštět a vyhodnocovat jakékoliv testy přes jednoduché rozhraní. Přínos by měl spočívat v tom, že se unifikuje spouštěcí a reportovací platforma a dosáhne se tak lepšího přehledu o dostupných testech a jejich výsledcích. Do budoucna tak půjde lépe specifikovat a navrhovat další testy bez potřeby hledání dílčích reportovacích nástrojů nebo spouštěcích skriptů. Obecně se takto otevřou dveře pro vylepšení kontroly kvality produktu Zoner Photo Studio X. Další cíl je poskytnout rozhraní k tvorbě testů, které se zaměřují na RAW soubory a potřebují předem překonvertovaný vstupní soubor, nad kterým bude prováděna kontrola. To znamená, že aby se dalo automatizovat testování nad zmíněnými RAW soubory, nebude třeba je vyhledávat a testovat manuálně, ale vytvořený testovací systém bude poskytovat rozhraní, díky kterému budou mít takové testy předem připravené vstupní soubory, nad kterými poté testy vykonají kontrolu.
Mimo návrh a implementaci testovacího systému pro Zoner Photo Studio se práce zaměřuje i na regresní blackbox testování obrazových výstupů modulu pro zpracování RAW souborů, aby byla zajištěna spolehlivost konverze z neobrazového formátu RAW do obrazových formátů, a to při zachování věrných barev a přijatelné rychlosti zpracování. To v profesionálním fotografickém softwaru hraje kritickou roli. Konkrétní návrh a realizace tohoto cíle je specifikována v sekci 2.4.2. Tento způsob testování je vysokoúrovňový a lze jej považovat za systémové testování, jelikož v podstatě kontroluje, zda se modul ZRaweneger chová, tak, jak od něj uživatel očekává. Tedy že převádí RAW soubory na obrazové formáty s věrným zachováním barev a že to dokáže v rozumném časovém úseku. Test na výkonnost tohoto modulu se zaobírá hlavně dobou nutnou pro převod RAW souboru. Kvalita obrazu a čas jsou dvě klíčové vlastnosti z pohledu uživatele, a tudíž jsou to hlavní kritéria pro chování systému.

2.3.1 V-model, aplikovatelnost v případě testování Zoner Photo Studia X a klasifikace navržených testů pro modul ZRawenger

V-model je proces vývoje softwaru, který může být považován za rozšíření vodopádového modelu. V-model ukazuje vztahy mezi každou fází životního cyklu vývoje a související fází testování. Místo lineárního pohybu dolů ve vodopádovém modelu jsou kroky procesu po kódovací fázi ohnuté nahoru, aby vytvořily typický tvar písmene *V* [8]



Obrázek 2.1: Struktura V-Modelu.

Po studiu tohoto modelu bylo shledáno, že tento model je neaplikovatelný na potřeby testování Zoner Photo Studia, jelikož se jedná o model, který se musí dodržovat již od počátku projektu a není vhodný pro dlouhodobé projekty. Problém je také v tom, že Zoner Photo Studio není od útlých začátků kompletně testováno a zpětné zavedení je relativně náročná záležitost, mimo jiné i kvůli přítomnosti *legacy* řešení. Vzhledem k tomu, že se testy, které jsem navrhl, soustředí na celkové specifikace systému, ve vztahu k V- Modelu by se daly klasifikovat jako systémové, jelikož testují již kompletní produkt.

2.4 Regresní testování, jeho definice a aplikace v testech konvertoru RAW souborů

Termín regresní testování je způsob opakovaného testování celku software, který byl jakkoliv změněn, aby se předešlo zanesení chyb do jinak funkčního programu nebo jeho modulu. Je to esenciální část drtivé většiny ustálených vývojových cyklů software. Ač tomu někteří vývojáři nemusí věřit nebo se tomu dokonce bránit, i po konfrontaci s důkaznými materiály, byť i minimální zásah do vzdáleného modulu systému může ovlivnit funkcionalitu jiné části softwarového systému. Právě tímto se zaobírá regresní testování.

2.4.1 Aplikace regresního testování

Regresní testování, jako takové, musí být z principu automatické. Pro automatizaci lze použít například některý otevřený testovací nástroj z široké nabídky, která je dnes k dispozici, popřípadě kombinaci více testovacích nástrojů. Příkladem může být Selenium[4], které pracují na bázi *capture and replay* , tedy *zaznamenej a přehraj*. Díky tomu se dá efektivně zautomatizovat například simulace uživatelského klikání myší na webové aplikaci nebo jiném uživatelském rozhraní. Do regresního testovaní ale náleží i jakékoliv obecné testy, které tester zahrne do množiny testovacích sad podléhající regresní kontrole. Tato množina však vyžaduje opakované vyhodnocení na základě úprav systému. Ty se dají dělit na čtyři varianty: korektivní, perfektivní, adaptivní a preventivní. Všechny tato kritéria ve výsledku vyžadují regresní kontrolu, i pokud přímo nemění obecný výstup nebo chování systému.
Do regresního testování zároveň není vhodné striktně zahrnovat všechny dostupné testovací sady. Paul Amman a Jeff Offutt ve své publikaci *Introduction to software testing* zdůrazňují, že regresní testy typicky mají být spouštěny přes noc, jelikož je časté, že jejich průběh zabírá množství výpočetního výkonu a s tím spojeného strojového času, který může být placený. Pokud budeme brát v potaz příklad firmy, kde se regresní testy spustí odpoledne po vydání aktualizace produktu, testy by měly být dalšího dne ráno dokončeny a připraveny k evaluaci testerem nebo vývojářem. Pokud by regresní testy přes noc nestihly doběhnout, může to rozhodit stabilitu vývojového procesu, jelikož nebude možné vydat produkt, jenž neprošel celým testováním a vývojový proces by tak byl narušen. Firma má poté dvě možnosti:

1. Navýšit výpočetní výkon serveru nebo stanice, na níž testy probíhají
2. Pravidelně optimalizovat testovací sadu a eliminovat marginální testy

První možnost bude ve většině případů vyžadovat navýšení výpočetního výkonu serveru nebo rozdělení testovacích sad na několik částí a ty spouštět paralelně na více strojích. To doprovází zvýšené náklady na běh testů. Záleží samozřejmě na vedení firmy, zda je toto přijatelné. Výhodou je, že nedochází k vynechání žádného testu, tudíž lze konstatovat, že pokrytí testy je maximální, jsou-li dobře navrženy. Nejedná se však o úplně optimální řešení, jelikož například testy, které regresivně kontrolují prokazatelně nezměněnou část nemusí být v dané iteraci vývojového cyklu spouštěny.
Druhá možnost nabízí optimálnější řešení, ale je nutné, aby tester správně vyhodnotil, zda je možné marginální testy opravdu vypustit. Testy lze obecně vynechat jen a pouze v tom případě, že každá část kódu ani knihoven a modulů v něm použitých, kterou takový test pokrývá, je prokazatelně nezměněná, což je vcelku náročná operace, obzvláště pokud se používají externí rozhraní nebo knihovny. V případě uzavřených knihoven bez veřejného zdrojového kódu je to navíc prakticky nemožné. V opačném případě se riskuje, že přeskočený marginální test selže (a je dobré vždy počítat s tím, že preskočený test se považuje za selhání) a do systému by se tak tiše propagovala chyba. [3]

2.4.2 Způsob regresního testování modulu pro převod RAW souborů

Regresní testy nad komponentou pro převod RAW souborů do obrazových formátů jsou v rámci této práce navrženy hlavně pro kontinuální kontrolu kvality výstupních obrazových dat. Tato problematika je blíže specifikována v kapitole 3. Cílem je, aby po změně v tomto

[4]Selenium je nástroj pro automatizaci testování webových aplikací. Dokumentace dostupná z https://www.selenium.dev/

modulu nebo v knihovnách, které používá, byly výstupy programu konzistentní pro všechny podporované typy RAW souborů. V praxi jsem se setkal například s tím, že při určitých parametrech RAW souboru může docházet k chybám hlavně v barvách fotografie, kdy se například nebe zobrazí fialové. Tomuto chování lze z velké části pomocí regresního testování zamezit. Přesný postup je poté k nalezení v kapitole 4.10.1. Zároveň je potřeba regresně ověřovat výkon modulu, protože je nutné, aby změna v kódu neovlivnila negativně rychlost převodu RAW souboru.

2.5 Testování založené na znalostech systému a volba typu v případě testování Zoner Photo Studia X

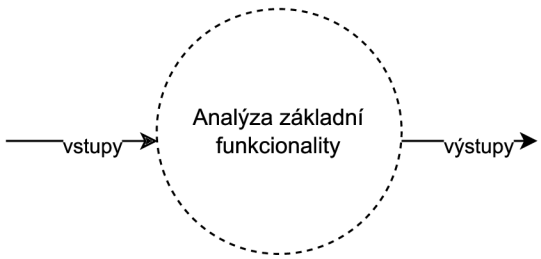
V testování založeném na znalostech o testovaném systému rozlišujeme dvě základní možnosti. Ty jsou na opačné straně spektra znalostí, které o systému máme.

1. Black box testování
2. White box testování

V této kapitole se zaměříme na jejich definici a obor uplatnění z hlediska fáze vývoje softwarového projektu. Také je zhodnotíme z hlediska uplatnitelnosti v rámci této práce a jejich vhodnosti pro testování Zoner Photo Studia X s respektem k fázi vývoje, ve kterém tento produkt je.

2.5.1 Testování způsobem black box

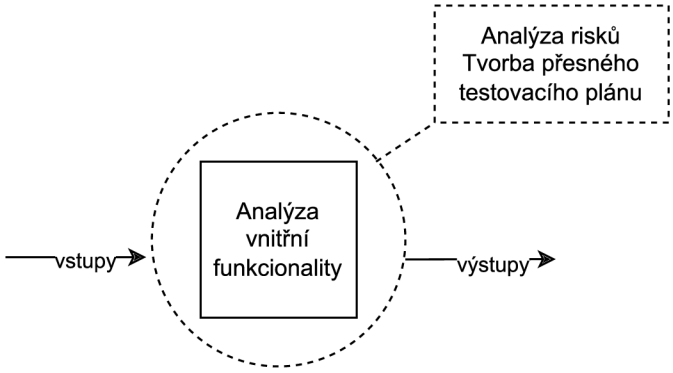
Testování způsobem *Blackbox*, také známé jako *testování řízené vstupy a výstupy* , je testování, ve kterém je pro nás testovaný systém v doslovném překladu *černá krabice* . To znamená, že se snažíme nebrat ohled na vnitřní mechanismy testovaného programu. Místo toho se soustředíme na hledání podmínek, ve kterých se program nechová tak, jaká je jeho specifikace. V tomto přístupu jsou vstupní data přímo derivována ze specifikace programu. Problémem tohoto přístupu je to, abychom zaručili stoprocentní korektnost chování programu, je potřeba vyzkoušet všechny možné kombinace vstupů. Tento přístup se jeví jako naprosto nevhodný pro řadu programů, například pokud bychom chtěli testovat způsobem blackbox překladač programovacího jazyka C++, množina všech vstupů by byla ekvivalentní s množinou všech myslitelných programů v jazyce C++. Taková množina je prakticky nekonečná a testování touto formou přirozeně nedává žádný smysl.[10]
Blackbox testování je vhodné použít v pozdních fázích vývoje softwaru, kdy jsou již funkce a vlastnosti softwaru plně implementovány a je třeba ověřit, zda splňují specifikace.



Obrázek 2.2: Reprezentace black box testování [7]

2.5.2 Testování způsobem white box

Tento způsob, na rozdíl od black box testování, povoluje testerovi prozkoumat interní strukturu programu a přístup ke zdrojovému kódu. Tester se v tomto případě zaměřuje na testování jednotlivých funkcí, procesů a algoritmů v testovaném systému. Díky této hloubkové znalosti mohou být snáze odhaleny chyby v kritických bodech algoritmu během raných fázích vývoje. Celkový cíl white box testování je odhalení implementačních chyb, aby mohly být opraveny dříve, než se vypropagují do vyšší fáze vývoje. Tam by je například black box testování již nemuselo zachytit. White box testování se provádí na základě kritérií pro pokrytí kódu a cest *control flow grafem*. Cílem je zjednodušeně pokrýt testy všechny možné výsledky sekcí či konstrukcí v kódu a zkoumat, zda v nich nevzniká chyba. Je ale důležité stále myslet na to, jaká je specifikace programu a neodklonit se od ní.[10]

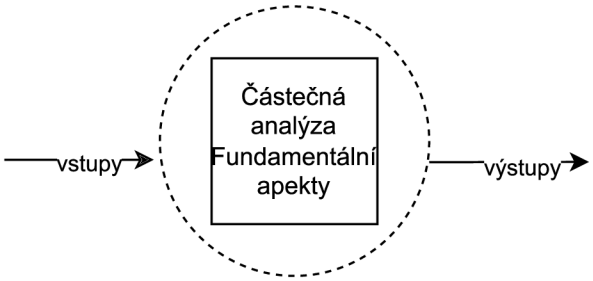


Obrázek 2.3: Reprezentace white box testování [7]

2.5.3 Vhodný způsob testování komponent Zoner Photo Studia X z pohledu znalostí o systému

Z výše uvedených informací vyplývá, že v rámci této práce bude vhodné se zaměřit na testování způsobem black box. Zoner Photo Studio je v pokročilé fázi kontinuálního vývoje, jak je popsáno v sekci 2.2. Pro zavádění testovací infrastruktury je tedy tato metoda vhodnější než white box testování. Dále již existují jednotkové testy, které píší sami vývojáři, a na které se dá spolehnout, jelikož jsou již delší dobu v produkci. Tyto testy budou s rozvojem tohoto projektu zahrnuty do navrženého testovacího systému. Dle Glenforda Myerse a jeho publikace *The art of software testing* by sice sám vývojář neměl testovat vlastní kód, jelikož jeho pohled na vlastní dílo může být neobjektivní[10], ale vzhledem k tomu, že tyto testy již existují a celkové pokrytí Zoner Photo Studia sahá za rozsah této práce, připadá metoda black box testování jako ideální pro testy navržené v rámci sady pro regresní kontrolu výstupů modulu ZRawenger.
Pro upřesnění, forma testování v této práci je založena na principu black box, ale prakticky

by se dalo spíše zařadit k hybridnímu grey box testování. Grey box testování je postup testování, kdy má tester povědomí o strukturálním fungování testovaného systému. Typicky má tester vědomosti o toku dat programem nebo o sekvenci vykonávání dílčích kroků. V tomto případě je to povědomí testera o krocích, které jsou esenciální pro převod RAW souboru do obrazového formátu. Problematika RAW souborů je vysvětlena kapitole 3



Obrázek 2.4: Reprezentace grey box testování [7]

Mezi výhody grey box testování oproti konvenčnímu black box nebo white box testování patří následující body:

- Tester spoléhá na definici rozhraní programu a na specifikaci, místo zdrojového kódu
- Tester cílí na kompletní pokrytí z pohledu zákazníka a nikoliv architekta systému
- Testování je přesnější, ale není ovlivněné znalostí zdrojového kódu

Na druhou stranu nevýhody tohoto postupu mohou být:

- Pokrytí testy je omezené
- Hodně cest zdrojovým kódem programu může zůstat neověřených
- Pokud již existují testy na nižší úrovni pro test navržený z pohledu grey box, mohou vznikat redundantní testy[7]

2.6 Výkonnostní testování

Většina programů má své specifické výkonnostní cíle, které lze určitým způsobem kvantifikovat a ověřovat. Může to být čas odpovědi u webových serverů nebo čas vykonání kalkulace u matematických algoritmů. Typickými metrikami u programů, kde hraji roli čas potřebný k vykonání operace bývají následující metriky:

1. Samotná rychlost vykonání operace
2. Zatížení jednotlivých jader procesoru
3. Spotřeba operační paměti
4. Četnost čtení a zápisu na disk

V případě testů modulu pro interpretaci RAW souborů je klíčová metrika čas od zahájením převodu RAW souboru do uložení obrazového formátu na disk. Díky počítání času až do zápisu na disk bereme v potaz i diskové operace. V rámci testování se tato metrika bude regresivně ověřovat, aby do ostrého vydání Zoner Photo Studia X nebylo v důsledku změn v algoritmech nebo externích knihovnách zaneseno zpomalení.
Další důležitou metrikou, která se bude kontrolovat stejným způsobem, je velikost alokované operační paměti. Tradičním způsobem pro regresní ověřování výkonnosti systému je stanovení takzvaných *benchmarků*, což je typicky reprezentace ideálního stavu, který je očekáván i v produkci. Pro dosažení optimálního benchmarku, se kterým se poté bude regresivně porovnávat výkon systému po aktualizaci, je potřebné nastavit ideální prostředí pro tvorbu těchto benchmark. Je třeba také zaručit, že tyto benchmark hodnoty budou shodné se specifikací programu a také s očekávaným výkonem po nasazení programu u zákazníka.[16] Detailní popis návrhu benchmark dat je popsaný v sekci 2.6

2.7 Zapojení testování do vývojového cyklu

Dle popisu vývojového cyklu v sekci 2.2 Zoner Pohoto Studio X je vyvíjeno ve dvou hlavních sprintech. Mimo tyto hlavní aktualizace jsou dále vydávany záplaty na odstranění chyb v programu a verze pro alfa a beta testování. Kanály pro správu vydávání jsou následující:

1. Alpha kanál, sloužící pro alpha testování (`ALPHA`)
2. Beta kanál, sloužící pro beta testování (`BETA`)
3. Hlavní produkční kanál (`STABLE`)
4. Testovací kanál, který slouží pro kontrolu záplat a hlavních aktualizací před ostrým produkčním vydáním do `STABLE` kanálu (`STABLE_TEST`)

Pro všechny tyto kanály jsou separátní instalační balíky, ke kterým je vedena evidence v interní databázi. Testování se bude provádět pro každý kompletní patch nebo aktualizaci na kanálech `STABLE_TEST`, `ALPHA` a `BETA`. Kanál `STABLE_TEST` je určený přímo k testování produkční verze před vydáním na kanál `STABLE`, tudíž tento bude hlavní pro spouštění celkového testování.
Testovací systém je navržen tak, aby se dal spouštět dvěma způsoby:

1. Požadavkem HTTP
2. Systémovým voláním instalátoru Zoner Photo Studia X na testovacím serveru.

Testování bude tedy vždy završením cyklu tvorby záplaty nebo aktualizace, ale navržený systém by měl být schopný spouštět i individuální testovací sady dle poskytnuté konfigurace v případě, že to bude některý z vývojářu potřebovat pro kontrolu nebo jiné účely. Kompletní popis návrhu je dostupný v sekci 4 a implementace v sekci 5

Kapitola 3

Problematika RAW souborů a modul určený k jejich zpracování v Zoner Photo Studiu X

Tato sekce se zabývá problematikou generování adekvátních obrazových dat z formátu RAW. Vědomosti v tomto ohledu jsou důležité pro navržení testovací sady konverzního modulu principem grey box, popsaném v sekci 2.5.3.

3.1 Obecný popis RAW souborů a jeho využití

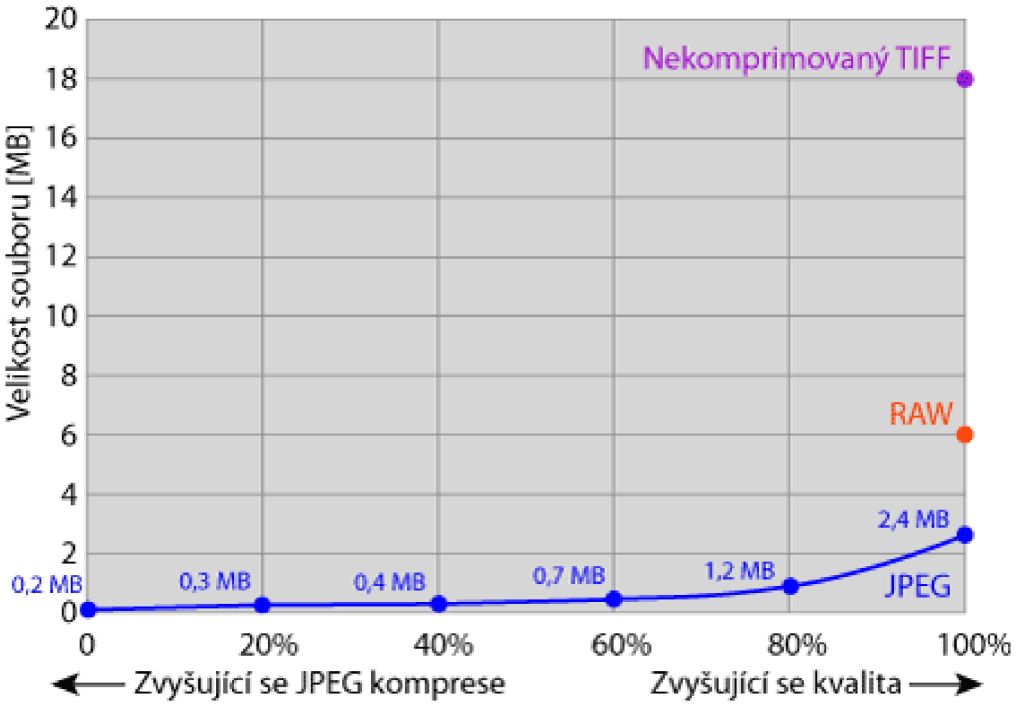
RAW je sohrnný název pro formát souboru, který obsahuje jen minimálně zpracovaná surová data ze senzoru digitálního fotoaparátu[1]. Často bývá nazýván také digitálním negativem, ač s negativním typem obrazu nemá nic společného. Podobně jako u předešlé technologie ve fotografii, negativu z filmového pásu, je RAW potřeba také určitým způsobem vyvolat, ale u RAW formátu se jedná o sekvenci výpočetních operací a nikoliv o chemickou reakci.[12] Formát RAW je určený pro profesionální fotografy, kteří potřebují, aby se v zachyceném snímku uchovalo co nejvíce dat z fotoaparátu. Tím lze dosáhnout mnohem detailnějších úprav v postprocesování ve fotosoftwaru a fotograf tak může například zachránit přepaly[2]. Z praktického hlediska je tak fotografování do RAW formátu pro fotografy velice výhodné, pokud mají zachytit nějaký klíčový moment a nemají více příležitostí snímek znovu pořídit. Například při fotografování momentů ze svatby nebo jiné unikátní události. V postprocesu mají v případě RAW souboru širší možnosti úprav fotografie, jelikož v něm jsou k dispozici přímo data ze senzoru fotoaparátu, s nimiž lze stále široce manipulovat. Pokud by místo RAW snímku fotograf pořídil snímek do obrazového formátu JPEG, fotoaparát vygeneruje již finální bitmapovou podobu snímku. I JPEG lze samozřejmě upravit, ale velké chyby, například v oblasti expozice, již nelze tak efektivně korigovat. Navíc JPEG soubory podléhají kompresi. Ve shrnutí jsou tedy RAW soubory vhodné, pokud plánujeme s fotografií ještě dále manipulovat. Nevýhodou může být na druhou stranu jejich velikost. Například u digitální MILC[3] kamery Fujifilm GFX 100S dosahuje jeden nekomprimovaný RAW velikosti okolo 230MB. Z tohoto důvodu nejsou snímky ve formátu RAW vhodné k archivaci

[1] V překladu anglické slovo RAW znamená surový

[2] Místa na fotografii, která se zobrazují příliš světlá. Příklad může být pixel RGB s hodnotami `(R=255, G=255, B=255)`, tedy maximální možná hodnota symbolizující čistou bílou barvu.

[3] Mirrorless interchangeable-lens camera - bezzrcadlová kamera s výměnným objektivem

a je lepší je převést do formátu JPEG či TIFF a tento soubor poté archivovat. To samé platí při distribuci snímku k zákazníkům, jelikož RAW soubor nelze jednoduše zobrazit. Zde se opět vyplatí formát JPEG nebo například komprimovaný TIFF či PNG. Porovnání velikosti formátů je zobrazené v obrázku 3.1 Popis generování této trojice formátů je poté popsán v sekci 3.2



Obrázek 3.1: Porovnání velikosti stejného souboru o rozlišení 3000 na 2000 pixelů ve formátech RAW, TIFF a JPEG s různou kompresí.[13]

3.1.1 Diverzifikace typů RAW souborů podle výrobců kamerových systémů

Jedním z hlavních a náročně řešitelných problémů pro software, jenž RAW soubory zpracovává a převádí, je fakt, že každý výrobce používá svoji vlastní formu RAW formátů. RAW formát není bohužel nijak standardizován a výrobci si každý podle svých potřeb, preferencí a použité technologie (senzoru) definovali vlastní RAW formáty. Rozdíly jsou značné, dokumentace k jednotlivým RAW formátům také není většinou dostupná. Je tak třeba používat programy určené pro konkrétní značku a často i pro konkrétní model. Výrobci navíc částečně pod vlivem technologického vývoje, částečně jako obchodní politiku, RAW formát model od modelu mění.[12]. RAW soubory od různých výrobců lze rozlišit dle koncovky toho kterého souboru. Seznam používaných formátů a výrobců je k nalezení v příloze A. Pro autory fotografických programů toto představuje značnou výzvu, jelikož s jednotlivými typy RAW souborů se mění i detaily ve způsobu jejich zpracování či jejich obsahu. Toto je způsobeno ze značné části snahou výrobců utajit interní mechanismy v rámci obchodní soutěže. Každý výrobce poskytuje vlastní software pro převod vlastních RAW souborů, který využívá detailní znalost způsobu generování dat ze senzoru fotoaparátu při

převodech. Někteří výrobci dokonce zachází tak daleko, že své RAW soubory šifrují, jako tomu bylo v minulosti u firmy Nikon[12], aby šly převádět pouze v dedikovaném softwaru výrobce. Problém těchto softwarů spočívá v tom, že jsou určeny hlavně pro převod a neposkytují kvalitní rozhraní pro efektivní postprocessing fotografií, navíc často trpí na nízkou uživatelskou přívětivost a podporu.

3.1.2 Formát DNG a pokusy o standardizaci

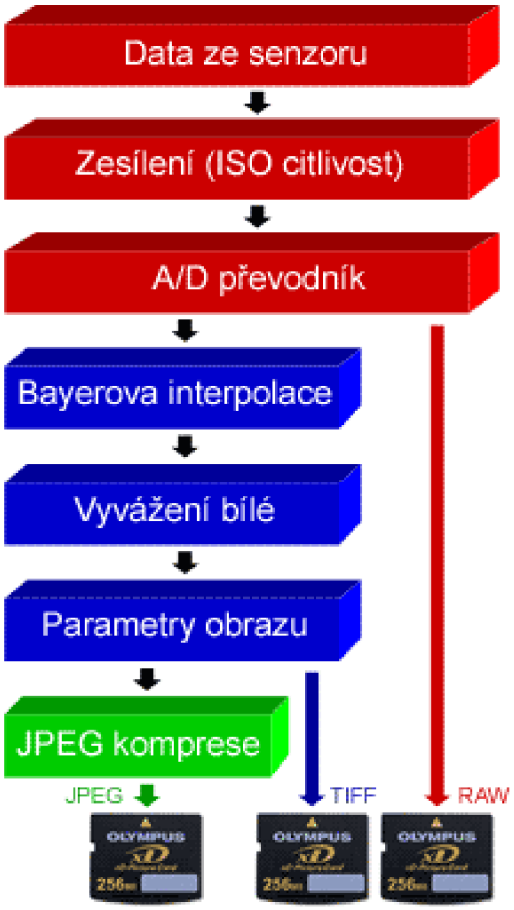
Nedostatek standardizovaného formátu pro soubory RAW vytváří dodatečnou práci i pro výrobce fotoaparátů, protože musí vyvíjet vlastní formáty spolu s softwarem pro jejich zpracování. To také představuje určitá rizika pro koncové uživatele. Formáty souborů raw fotoaparátů se liší od fotoaparátu k fotoaparátu, i u těch od stejného výrobce. Není neobvyklé, že výrobce fotoaparátů ukončí podporu pro formát souborů RAW starších modelů. To znamená, že uživatelé nemají záruku, že budou moci v budoucnu otevřít archivované soubory raw. Nejvýznamnější pokus o standardizaci učinila firma Adobe, která definovala svůj otevřený a standardní formát DNG (Digitální Negativ) a nabídla ho výrobcům digitálních fotoaparátů a jiných zařízení.[1]
Digitální negativ je otevřeně publikovaná specifikace souborů raw, která ukládá nezpracované obrazové body zachycená snímačem digitálního fotoaparátu před převodem na formát JPEG nebo TIFF, spolu se standardními metadaty EXIF, datem, časem a informacemi o použitém fotoaparátu a jeho nastavení. Tento formát je volně dostupný ostatním dodavatelům softwaru a hardwaru. Formát DNG někteří z výrobců digitálních kamer dokonce přijali jako hlavní výstupní RAW svých produktů, například společnosti Sigma a Pentax.[12]. Společnost Adobe navíc vyvinula veřejně dostupný nástroj Adobe Digital Negative Converter[4], který poskytuje rozhraní pro převod téměř všech typů RAWů do formátu DNG. Výhoda tohoto formátu pro tvůrce fotosoftwaru je, že DNG poskytuje unifikované rozhraní pro převod do obrazových formátů, narozdíl od RAW souborů, které se od sebe navzájem liší. Z hlediska zpracování RAW souborů je tedy výhodné použít převod do DNG jako mezikrok při převodu RAW souborů různých výrobců do obrazových formátů ve fotografických editorech třetích stran. Zoner Photo Studio X tento mezikrok úspěšně implementuje. Bližší popis je k nalezení v sekci 3.5

[4]Informace dostupné z https://helpx.adobe.com/cz/camera-raw/using/adobe-dng-converter.html

3.2 Způsob ukládání výstupů ze senzoru fotoaparátu souvislosti s nejpoužívanějšími formáty

Dle popisu vysoké diverzifikace RAW souborů mezi jednotlivými výrobci kamerových systému v sekci 3.1.1 se dá předpokládat, že každý výrobce bude mít i svůj vlastní postup ukládání RAW souborů. Tyto způsoby jsou, jak bylo řečeno, neveřejné. Existuje ale obecný postup při generování souborů formátů JPEG, TIFF a RAW v samotném fotoaparátu.

- JPEG z fotoaparátu již má na rozdíl od RAW souboru zprocesované všechny dílčí operace pro převod signálu ze senzoru do obrazových dat. Navíc se zde používá JPEG komprese.
- TIFF je nekomprimovaný, ale obsahuje již zprocesovaný obraz, podobně jako JPEG.
- Obrazová data v RAW souboru obsahují pouze digitalizovaná data ze spojitého signálu senzoru.



Obrázek 3.2: Diagram sekvence operací ve fotoaparátu při vytváření a ukládání souborů ve formátu JPEG, TIFF a RAW na paměťové médium.[14]

3.2.1 Datové sekce RAW souboru

RAW soubor typicky obsahuje několik serializovaných bloků dat, z nichž každý má jiný účel a nese jiné informace o snímku pořízeném digitálním fotoaparátem. Jednotlivé segmenty a jejich uspořádání se může u souborů RAW různých výrobců lišit. Bloky dat, které lze považovat za obecné, jsou následující:

1. Metadata: Data o datech, respektive zachyceném snímku. Tento segment obsahuje například datum pořízení, lokaci pořízení v GPS souřadnicích, hodnota ISO[5] a další. Tyto informace jsou uchovány v EXIF [6] formátu uvnitř RAW souboru.

2. Obrazová data: Tento blok obsahuje samotnou obrazovou informaci zachycenou senzorem fotoaparátu v neupravené podobě. Obsahuje všechny záznamy barev, tonů a kontrastů získaných v průběhu snímání.

3. Náhledová data: RAW typicky obsahuje i zmenšený náhledový JPEG soubor, který se používá pro zobrazení náhledu, například při listování seznamem RAW souborů. Díky tomu není potřeba RAW kompletně konvertovat do obrazových dat, pokud v dané situaci stačí uživateli zobrazení náhledu. Tomuto JPEGu se často říká *embedded JPEG*, v překladu *zabudovaný JPEG*.

4. Tabulková data a další záznamy, které se používají pro korektní interpretaci surových dat při převodu do obrazových formátů.

Tato data a struktura jejich záznamu se typicky liší podle výrobce a proto je důležité RAW soubor i korektně načítat a dělit na sekce. Toto v Zoner Photo Studiu řeší externí otevřená knihovna.

3.3 Obecný postup převodu RAW souboru do zobrazitelných formátů a sekvence dílčích operací

Pipeline[7] převodu RAW souborů se může u každého producenta fotosoftwaru lišit, ale obecně opět exitují základní kroky, které jsou nutné. V seznamu níže jsou popsány kroky, které jsou obecně prováděny i v konverzním modulu Zoner Photo Studia X.

1. Načtení metadat RAW souboru, zjištění jeho typu a korektní načtení bloků dat do paměti. Další kroky se provádí nad obrazovými daty

2. Normalizace obrazu

3. Demozaikování

4. Korekce související s objektivem včetně chromatické aberace

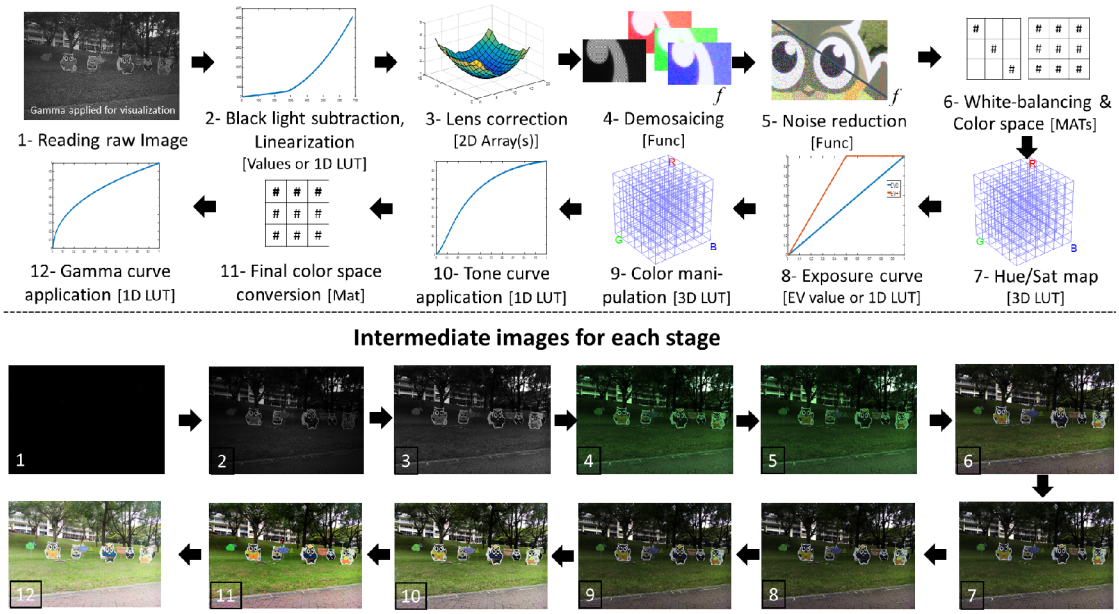
5. Redukce šumu

6. Vyvážení bílé a načtení do interního barevného prostoru pro zpracování, například *ProPhoto RGB*.

[5] ISO hodnota je nastavení senzitivity snímacího senzoru.
[6] Exchangeable Image File Format: https://exiftool.org/TagNames/EXIF.html.
[7] Tok úloh, v překladu z anglického jazyka.

7. Úpravy pomocí expozičních křivek. Používá se look-up tabulka specifická pro fotoaparát.
8. Mapování odstínu a sytosti pomocí look-up tabulky[8] specifické pro fotoaparát
9. Procesy úprav barev a prokládání tonální a gamma křivkou. Detaily postupu bývají specifické pro jednotlivé RAW převodníky. Opět se používá look-up tabulka specifická pro fotoaparát.
10. Převod do výstupního barevného prostoru, například sRGB
11. Aplikace finálních efektů a export.

Výsledkem tohoto procesu je vyrenderovaný bitmapový obraz, který lze buď exportovat pro uložení, nebo dále upravovat. Úpravy v testovaném systém Zoner Photo Studio X lze provádět i nad samotnými RAW soubry. Pro zobrazení takové úpravy je potřeba projít tuto pipeline znovu s aktualizovanými údaji o úpravách. Pipeline se však neprochází celá, ale pouze od toho kroku, kde se daná úprava aplikuje. Pro optimalizaci tohoto procesu se využívá průběžné ukládání do vyrovnávací paměti.



Obrázek 3.3: Pipeline transformace a úprav surových dat z RAW souboru do odpovídajícího bitmapového souboru. Konkrétní postup na obrázku je specifický pro práci [5], najdou se tedy rozdíly mezi seznamem výše, který je generalizovaný dle postupu používaném v Zoner Photo Studiu.

3.3.1 Bayerův filtr a demozaikování

Proces v kroce 4 v sekci výše, tedy demozaikování, je princip získávání barev z jinak černobílých dat zachycených na senzoru fotoaprátu. CMOS senzor, jenž je přítomný ve většině

[8]Look-up tabulka je statická tabulka hodnot, které se používají při transformaci vstupních dat na výstupní.

moderních digitálních fotoaparátů, je citlivý pouze na intenzitu světla, nikoliv na barvy. Proto se před něj typicky umísťuje Bayerův filtr, jenž je zástupcem paradigmatu CFA.[9] Bayerův filtr byl vynalezen již v roce 1976 Brucem Bayerem ve firmě Kodak. Jedná se o matici složenou z mikroskopických barevných filtrů modelu RGB, jenž se v matici pravidelně opakují. Na sekce CMOS senzoru pokryté tímto filtrem tak dopadají pouze jednotlivé barevné složky světla. Neznamená to, že CMOS filtr tak přímo zachycuje barvu, jelikož jeho výstup je stále pouze intenzita světla. V každé podmatici Bayerova filtru s rozměry 2x2 jsou umístěny dva filtry pro kanál zelené barvy a jeden pro červený a zelený kanál. Tomuto rozložení se říká RGGB, právě kvůli zastoupení jednotlivých barevných kanálů ve filtru. Je vybráno kvůli senzitivitě lidského oka, které je nejvíce citlivé na zelenou barvu a výsledný obraz tak musí tomuto faktu podléhat, aby pro lidi působil přirozeně. Zelené barvy je tak vyfiltrováno méně. Rovnice složek RGB modelu pro relativní jas, Y, z níž toto vychází, je následující:

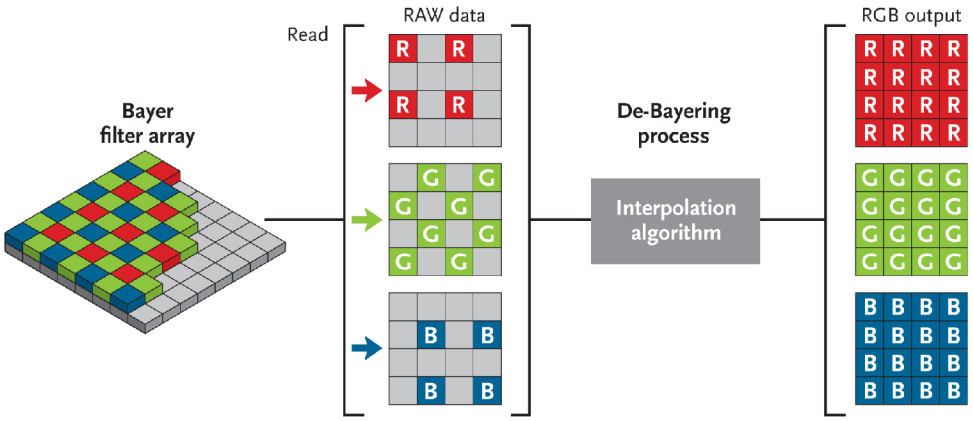
$$Y = 0.2126 * R_{lin} + 0.7152 * G_{lin} + 0.0722 * B_{lin}$$

Při demozaikování se poté z dat senzoru vytvoří matice podle použitého Bayerova filtru pro každý barevný kanál. Tyto matice nesou hodnoty jednotlivých barevných kanálů pouze na pozicích, na kterých byl zaznamenán příslušný barevný kanál. Ostatní pozice zústavají prázdné, respektive vynulované. Pokud bychom tyto tři matice sečetli, dostali bychom již barevný obraz, ale na každém pixelu, respektive pozici v matici, by byla zahrnuta pouze jedna barevná složka v určité hodnotě. Tento jev je ilustrován v kroce 3 obrázku 3.5. Proto je potřeba tyto matice interpolovat, aby se do každého pixelu zanesla hodnota pro všechny barevné kanály a bylo docíleného hladkého obrazu s co nejbližší podobou realitě. Některé typické interpolační algoritmy jsou například *Variable Number of Gradients*, *Adaptive Homogeneity-Directed* nebo *Aliasing Minimization and Zipper Elimination*. [6] V praxi se mimo Bayerova filtru používá i jiné uspořádání matice filtrů před senzorem fotoaparátu, například společnost Fujifilm používá filtr X-Trans. Ten, stejně jako Bayerův filtr, podléhá paradigmatu CFA, ale není pravidelný a využívá jiné uspořádání polí ve filtru. Z pohledu testování procesu zpracování RAW souborů toto opět přináší další komplikaci a riziko vzniku chyb.

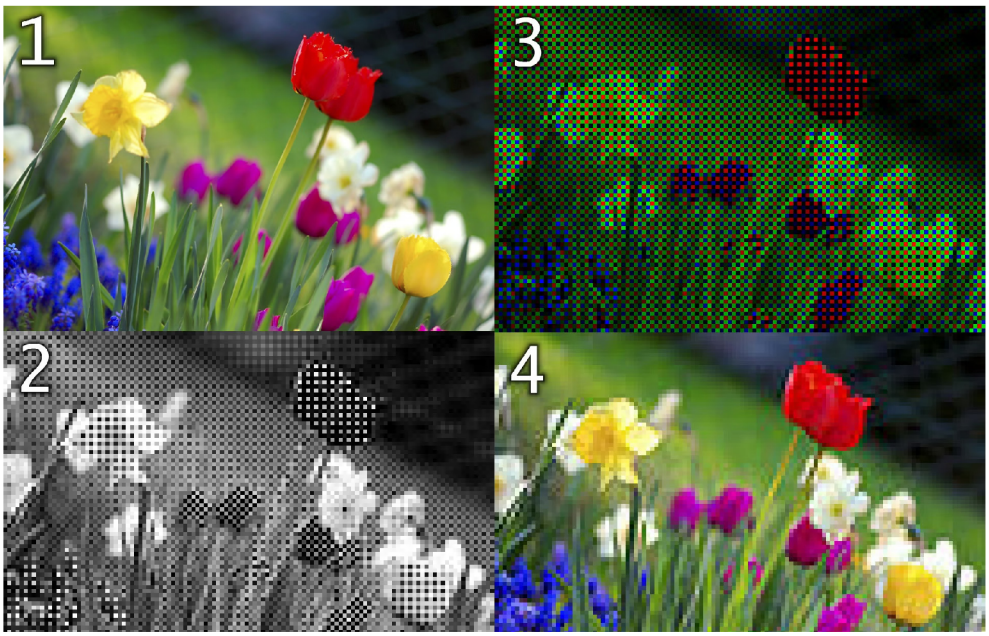
3.3.2 Příklady chyb při převodu RAW souborů

Vzhledem k relativně vysoké komplexitě a množství kroků v převodu RAW souborů do obrazových formátů lze předpokládat, že bez důsledného testování na úrovni kódu se mohou objevit ve výsledném obraze chyby. Problém při testování těchto chyb se dá dělit na jednotkové testy algoritmů v jednotlivých krocích převodu, a na kontrolu finálních obrazových výsledků. Tato práce se soustředí primárně na regresní ověřování nových chyb v již funkčním systému, tudíž se po validaci benchmark souborů ze všech podporovaných fotoaparátů bude kontrolovat obrazová shoda souborů převedených v nových verzích programu.
Níže jsou nastíněny některé typické chyby, které lze pomocí tohoto testování detekovat. Artefakty jsou řazeny do souvislosti s algoritmem kroku převodu. Je třeba podotknout, že artefakty mohou být viditelné pouze po velkém přiblížení obrazu, což je jeden z důvodů, proč byl v rámci regresního porovnávání obrazové shody vybrán přístup kontroly hodnot jednotlivých pixelů. Toto je detailně popsáno v sekci [[**SEKCE NA PER PIXEL**]]

[9] Color filtering array, jde o paradigmat překrývání snímačů barevným filtrem pro zachycení informací o barvách.



Obrázek 3.4: Postup při demozaikování surových dat ze senzoru fotoapáratu. Dostupné z https://www.skyandtelescope.com/wp-content/uploads/Redeeming-3.jpg

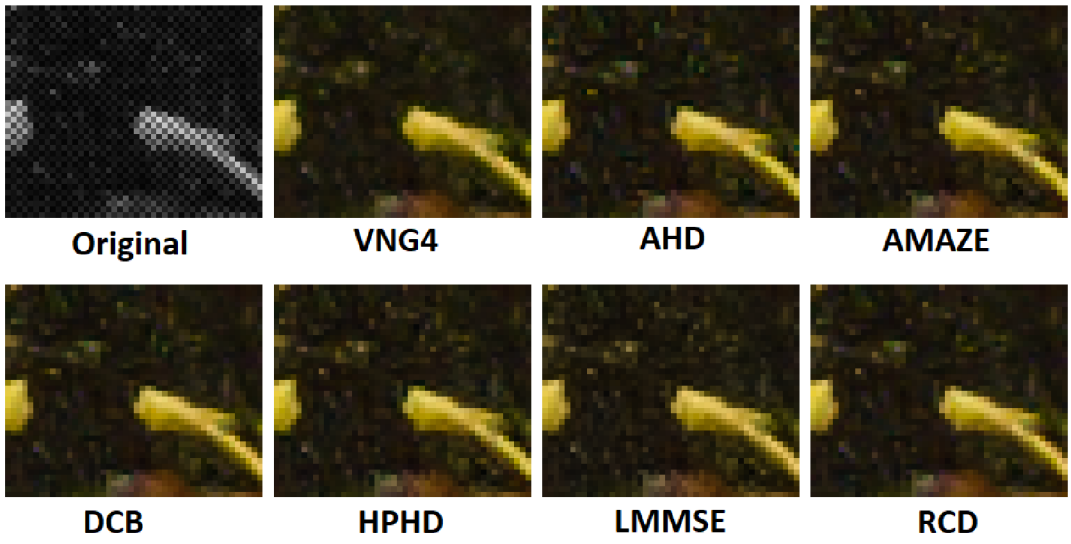


Obrázek 3.5: Vizualizace postupu demozaikování na příkladu. Krok 1 je původní zachycovaná scenérie, krok 2 jsou surová data ze senzoru, krok 3 jsou matice jednotlivých barevných kanálů a krok 4 je výsledný obraz po interpolaci těchto matic. Dostupné z blogu Hisour: https://www.hisour.com/bayer-filter-24546/.

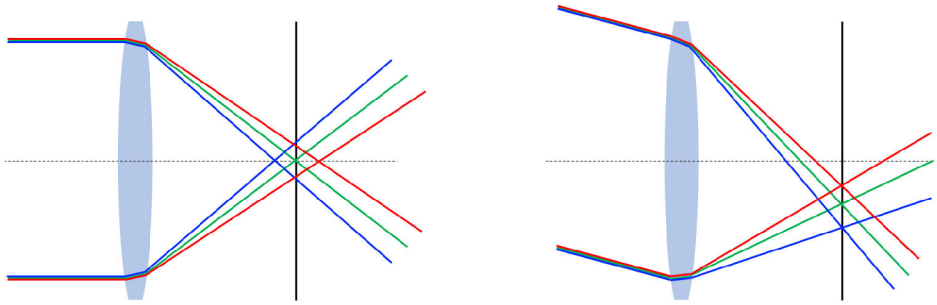
Artefakty způsobené chybou korekce objektivu

- **Chromatická aberace** vzniká z důvodu ohybu světla různých vlnových délek na čočce objektivu. Jedná se o přirozený jev, který ale lze algoritmicky redukovat. Prakticky nevadí, když není v převodu nijak opravena, jelikož ji lze v Zoner Photo Studiu X později odstranit aplikací LCP[10] profilu a případně manuální korekcí. V testování je ale žádoucí jakoukoliv změnu detekovat.

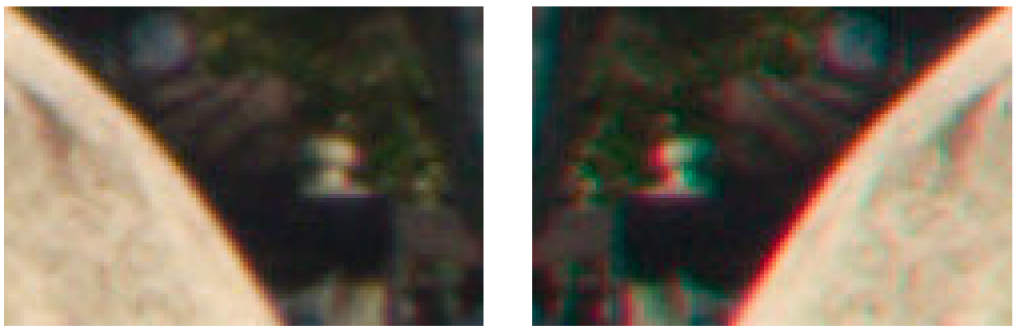
[10]Lens correction profile, v překladu profil úprav objektivu.



Obrázek 3.6: Vizualizace výsledků různých interpolačních algoritmů určených pro CFA. [6]



Obrázek 3.7: Znázornění vzniku chromatické aberace v důsledku nedokonalého lomu světla různých vlnových délek na čočce objektivu.[11]



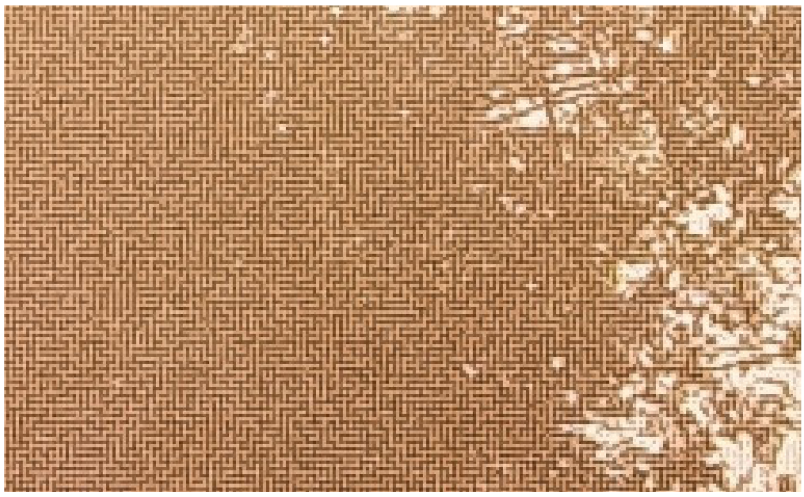
Obrázek 3.8: Srovnání snímku s chromatickou aberací (vpravo) a bez chromatické aberace (vlevo).[11]

Artefakty v úrovni demozaikování

- **False color artefakty** typicky vznikají na hranách odlišných odstínů v důsledku nedokonalostí interpolačního algoritmu.

Obrázek 3.9: Příklad chyby false color. Ve vrchní části oranžového objektu si lze všimnout nenáležící modré barvy.[15]

- **Moiré artefakty** mohou vznikat na větších plochách se stejným odstínem opět v důsledku nedokonalostí interpolace barev. Typicky tvoří tvary podobné bludišti nebo mřížky.



Obrázek 3.10: Dobře viditelná struktura bludiště. Chyba pravděpodobně vznikla v důsledku problému demozaikovacího algoritmu v programu Darktable. Dostupné z fóra *photo.stackexchange*: https://photo.stackexchange.com/questions/93613/why-is-my-camera-drawing-labyrinths-on-my-photos

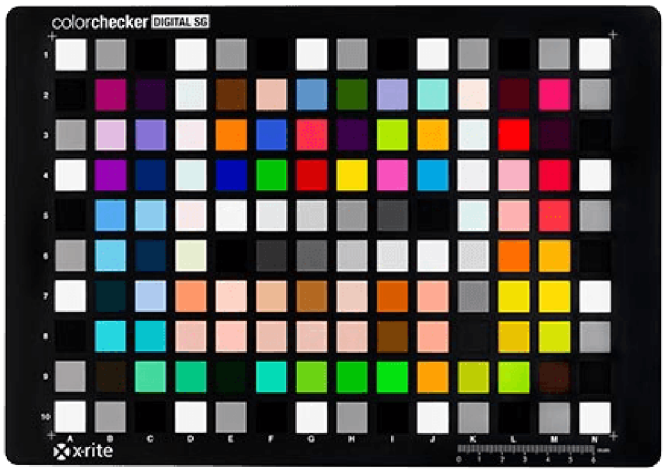
3.4 DCP Profily

DCP, zkráceně Digital Camera Profile, je formát souboru, který se používá pro uložení barevných profilů korekcí používaných při převody RAW souborů do standardních barevných prostorů. DCP profil se vztahuje právě k jednomu modelu fotoaparátu, pro který uchová

informace o korekcích. Obsahuje následující informace, které jsou specifické pro každý fotoaparát a jeho senzor:

- Informace o kalibraci iluminantu
- Sada barevných matic
- Dvě look-up tabulky pro mapovaní sytosti a odstínu barev v krychli RGB modelu
- Profilová tonální křivka

Přesný obsah lze dohledat v [1] Pro tvorbu DCP profilů existují nástroje a lze je s dostatečným úsilím vytvořit i pro profesionální užití bez nutnosti poznatků o interním fungování fotoaprátů. Je k tomu však potřeba speciální vybavení a přístup k fotoaparátu, pro nějž chceme profil vytvořit. Vytvoření DCP profilu zahrnuje hlavně měření barev pomocí speciálních kalibračních terčů, které jsou fotografovány v exaktních laboratorních světelných podmínkách. Z naměřených dat jsou vytvořeny potřebné údaje a z nich pak DCP profil. Výhoda vlastních DCP profilů je možnost přesnější reprodukce barev. Pro tvorbu DCP profilů potřebných pro Zoner Photo Studio X se využívá placený nástroj třetí strany. Designer[11] a různé kalibrační terče.



Obrázek 3.11: Standardní kalibrační terč se 140 poli, který se používá i v interním procesu tvorby DCP profilů pro Zoner Photo Studio X. Tržní cena těchto terčů je okolo 370 amerických dolarů.

3.5 Integrovaný modul pro zpracování RAW souborů Zoner Photo Studia X

Zoner Photo Studio má vlastní integrovaný modul pro převody, úpravy a další potřebnou manipulaci s RAW soubory, který je v rámci této práce testován. V tomto procesu se využívá několik kroků, které nebyly popsány v obecném postupu při převodu v sekci 3.3. Hlavním

[11] https://www.lumariver.com/

rozdílem je, že se RAW soubor převádí nejdříve do formátu DNG, čímž je zaručena lepší uniformita pro algoritmy, jenž interpretují data ze souboru.

Celkový tok dat při převedení RAW souboru začíná načtením datových bloků popsaných v sekci 3.2.1. V rámci načítání dat se načte i informace o typu RAW souboru a fotoaparátu, pomocí něhož byl snímek vytvořen. K tomuto kroku je využita externí knihovna. Nyní se dostáváme do prvního větvení, kde je potřeba vybrat program, jenž provede konverzi do formátu DNG. Rozhodování probíhá na základě existence interních DCP profilů fotoaparátu, které jsou tvořeny na míru pro tento modul Zoner Photo Studia X. Způsob tvorby DCP profilů je popsán v sekci 3.4.

- Pokud je převáděn RAW z fotoaparátu, pro který existuje interní načítací profil DCP, převod provede interní modul.
- Pokud není k dispozici interní profil, provede se převod pomocí volně dostupného programu Adobe Digital Negative Converter, který je instalován s vlastními DCP profily.
- Pokud není možné využít ani jednu z předchozích variant a RAW je možné převést zastaralým legacy nástrojem, který se používal před aktuálním interním načítáním, je tak provedeno.
- Pokud neplatí ani jedna z předchozích možností, RAW není převeden.

Jakmile je k dispozici DNG ekvivalent převáděného RAWu, je tento načten pomocí knihovny Adobe DNG SDK[12]. Tato knihovna poskytuje rozhraní pro všechny potřebné operace nad daty v DNG souboru. Další mechaniky interního RAW interpretu jsou na ní založeny, ale výsledná implementace se od této knihovny liší. Důvodem jsou různé optimalizace a vylepšení pro získávání kvalitnějšího obrazu než konkurence, která by toto SDK používala přímo. S pomocí této knihovny je provedeno načtení DNG do paměti a provede se standardní sekvence zpracování, popsaná v sekci 3.3 a vizualizovaná v obrázku 3.3.

[12]SDK znamená software development kit, tedy sada nástrojů pro vývoj softwaru.

Kapitola 4

Návrh automatického testování Zoner Photo Studia X

Tato sekce se věnuje strukturálnímu návrhu testovacího systému dedikovaného pro Zoner Photo Studio X. Návrh celku testovacího systému vychází z nastudovaných poznatků o nástroji Zoner Photo Studio X v kapitole 2. Implementace regresních a výkonnostních testů nad modulem pro převod RAW souborů je poté popsána v sekci **??**.

Tento projekt se celkově vyvinul z kdysi používaného manuálního testování před vydáváním nových verzí, kdy byl jeden ze zaměstnanců pověřen manuálně zkoušet převody RAW souborů dle daných metodik. Vizuálně kontroloval chování systému, včetně manuálního měření času a dalších aspektů souvisejících s tímto modulem. Tato metoda ověřovaní kvality je velmi nepraktická, pokud bereme v potaz, že tyto testy jsou řízené vstupy. Pokud by tedy měl nějaký zaměstnanec korektně otestovat modul pro převod RAWů, musel by testovat celou validní vstupní množinu tohoto modulu. To v praxi znamená kombinaci všech RAW souborů se všemi kombinacemi parametrů ze všech podporovaných fotoaparátů. Taková množina je rozsáhlá, její testování ručně by tedy zabralo značné množství času a finančních prostředků. Navíc z tohoto procesu nelze vyloučit lidskou chybu a tak se jeví jako značně nefektivní. Toto manuální testování bylo při mém nástupu do firmy již zrušeno. V důsledku tendencí alespoň nějak modul pro konverzi RAWů prakticky testovat, padl návrh na jeho opětovné zavedení, ale ten se později přetavil v tento testovací projekt.
Cílem je tedy navrhnout systém, který posune testování ve firmě kupředu a zároveň vytvoří unifikovanou platformu pro implementaci budoucích testů.

4.1 Vytyčení firemních požadavků na testování

V počátcích tohoto projektu bylo přirozeně potřeba vytyčit jeho cíle. Finální výsledky výzkumu požadavků a potřeb společnosti Zoner jsou uvedeny v kapitole 2, celkový proces vytyčení těchto cílů byl však relativně zdlouhavý. Bylo to způsobeno obecně relativně vágními požadavky na testovací systém, protože unifikované testování nebylo ve firmě doposud pořádně zavedeno. Vývojaři ve firmě jsou často zabraní do svého vlastního úkolu, jenž mají zadaný a věnují se mu dlouhodobě. Pod úkolem si lze představit údržbu a rozvoj například

uživatelského rozhraní, matematických metod pro zpracování obrazu, modul zpracování RAW souborů nebo třeba relativně nový modul pro zpracování videa. Vývojář zodpovědný za danou oblast se často i sám věnuje svému internímu testování kritických bodů modulu a v případě objevení chyby uživatelem zodpovídá za její odstranění.

4.2 Specifikace

Přímé firemní specifikace vycházejí ze zmíněného manuálního testování. Úkolem je navrhnout a implementovat regresní testy modulu určeného pro převody RAW souborů. Ty by měly kontrolovat obrazovou shodu a čas konverze mezi poslední verzí Zoner Photo Studia v kanálu `STABLE` a nově vydanou verzí na kterémkoliv z kanálů `STABLE_TEST`, `ALPHA` a `BETA`. Mimo tuto primární kontrolu měla být navržena testovací sada pro kontrolu rozměrů převedeného RAW souboru a připraveno unifikované prostředí pro návrh dalších testů v této oblasti. Zároveň je požadováno, aby testovací systém fungoval na nativní platformě Zoner Photo Studia X, tedy operačním systému Microsoft Windows.

4.2.1 Rozšíření specifikace pro testovací systém

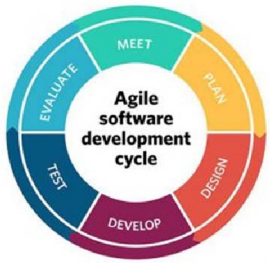
Mimo tyto přímé specifikace jsem se zaměřil na návrh a implementaci systému, který by byl schopný poskytnout rozhraní pro případnou integraci existujících testů z jiných modulů programu, které tvoří sami vývojáři. Dle [10] není vhodné, aby byl vývojář zároveň testerem svojí práce, ale vzhledem k omezené pracovní síle vývojářů a velikosti projektu jsou i tyto testy lepší než žádné. Jejich integrace do jednotného systému by mohla pomoci pokročit v oblasti zaručení kvality prodktu Zoner Photo Studio X. Popis možnosti zahrnutí těchto testů je popsán v sekci 4.6

4.3 Vývojový model uplatněný při návrhu a tvorbě systému

Během návrhu a implementace systému bylo směřováno k využití *agile* metodik. Největší výhodou této metodiky je její flexibilita a přizpůsobivost měnícím se požadavkům. Další výhodou je fakt, že projekt vyvíjený touto metodikou přináši hodnotu již během vývoje, na rozdíl od lineárního přístupu, kdy projekt přináší zákazníkovi hodnotu vždy až po jeho dokončení. Tento model je navíc vhodný pro projekty, kde není třeba striktně následovat interní firemní postupy a proces tvorby je flexibilní.[9]. Celková tvorba projektu byla rozdělena na dva větší sprinty[1]. Sprinty typicky trvají jeden až čtyři týdny, ale z důvodu potřeby výzkumu a samostatné práci na implementaci bylo nutné je prodloužit. Na konci obou sprintů vznikla funkční verze testovacího systému. První prototypovací sprint byl započat v červenci 2022 a dokončen v lednu 2023, druhý sprint byl poté dokončen v dubnu téhož roku. Na konci druhého sprintu stojí aktuálně nasazený pokročilý prototyp testovacího systému. Díky rozdělení na sprinty bylo možné obratně upravovat specifikace systému během jeho konstrukce. Například při reevaluaci databázového systému a přechodu z SQL databáze z prvního sprintu na NoSQL databázi ve druhém.

[1] sprint je termín pro určité časové období, ve kterém se vývoj soustředí na určité cíle a na jeho konci dochází k vyhodnocení výsledků. Používá se často v přístupu *scrum*.

Pro uvedení do perspektivy – při tvorbě testovacího systému pro Zoner Photo Studio X neexistoval přesně předepsaný postup a tak součástí tohoto projektu byl i výzkum možností testování. Pokud by tedy například uprostřed procesu tvorby tohoto systému vznikl urgentní požadavek na testy nad jinou komponentou, bylo by možné díky agile přístupu projekt přeorientovat uprostřed vývoje, bez nutnosti začínat znovu. Zároveň bylo potřeba opakovaně validovat směrování tvorby testů a případně je flexibilně upravovat, v čemž agile přístup napomáhá.



Obrázek 4.1: Ilustrace opakujícího se cyklu agilního přístupu k vývoji. Převzato z [9]

4.4 Uplatnění modulárního návrhu

V rámci návrhu samotné struktury systému jsem se rozhodl využít modulárního návrhu v kombinaci se základním využitím objektově orientovaného programování. Modulární návrh spočívá v dekompozici celého systému na nezávislé moduly. Tento přístup je vhodný hlavně díky možnosti moduly opakovaně použít při různých cílích spuštění programu. Zvyšuje se díky tomu přenositelnost kódu, přehlednost systému a údržba i testování se stávají snažšími, jelikož lze pracovat pouze nad izolovaným modulem. Zároveň tak systém dosahuje vysoké flexibility. Programovací jazyk Python, který byl zvolen jako hlavní technologie pro programování, je modulárně založený, tvorba modulů a vazeb mezi nimi je velice intuitivní proces. Pro vizualizaci souvislosti jednotlivých modulů byly vytvořeny dokumentační diagramy. Ty jsou dostupné v příloze B a celkový popis architektury je dostupný v secki 4.8.2

4.5 Návrh tříd testů v systému a generalizace společných kroků

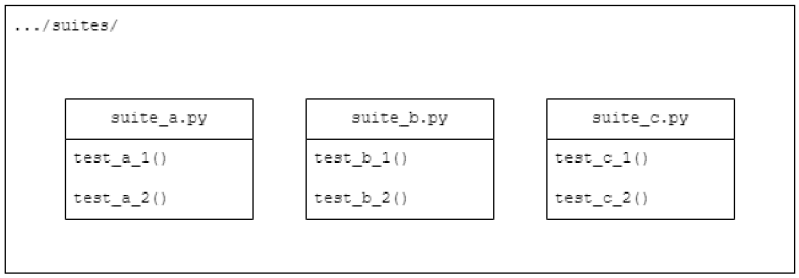
Dle specifikací v sekci 4.2 byly v rámci této práce vytvořeny regresní a výkonnostní testy, jenž jsou přímo integrovány do testovacího systému a dále jedna jednoduchá testovací sada pro kontrolu rozměrů konvertovaných RAW souborů. Tato testovací sada využívá rozhraní testovacího systému pro obecné testy, jenž vyžadují na vstupu konvertovaný soubor. Rozdíl je v tom, že integrované regresní testy se chovají jako samostatná sada, která aktivně interaguje s moduly v systému a dokonce je jeden z modulů, `benchmark`, dedikovaný přímo pro tuto testovací sadu. Testovací sada pro rozměry, pojmenovaná pracovně `suite_size`, využívá rozhraní systému, které lze využít pro implementaci testů, jenž nevyžadují přímou interakci s vnitřními mechanismy systému, ale stačí jim předkonvertovaný RAW na vstupu. Ve výsledku byly navrženy čtyři třídy testů.

1. Integrované regresní testy obrazových dat a výkonu RAW převodníku
2. Obecné testy vyžadující předem konvertovaný RAW soubor
3. Obecné testy bez vstupů
4. Integrované testy samotného převodu RAW souborů

Třída 1 přímo vyhovuje specifikaci na testování zadané školou. Tyto testy jsou tedy v podstatě požadovaným výsledkem této práce. Testy s vstupním souborem i základní testy poté využívají rozhraní systému a řadí se do rozšířené specifikace systému ze sekce 4.2.1 . Třída 4 je také integrovaná do systému a jedná se o způsob testu, jenž ověřuje, zda je RAW vůbec převeden a pokud ano, v jakém čase toho bylo docíleno.

4.5.1 Návrh rozhraní pro unifikovanou implementaci testů tříd 2 a 3

Rozhraní bylo navrženo tak, aby automaticky obsluhovalo testovací sady obsahující testy náležící do tříd 2 a 3 definovaných v seznamu výše. Konečným údělem tohoto rozhraní systému je, aby co nejvíce usnadnilo tvorbu a spouštění testů.
Pro testy třídy 2 je to zajištěno pomocí poskytování předem překonvertovaných RAW souborů, příjmání různých výsledků testů ve formátu množiny párů klíč-hodnota, evidencí výsledků a automatického hlášení výsledků těchto testů.
Pro testy třídy 3 rozhraní přijímá pouze binární výsledek, tedy informaci zda test prošel, nebo ne. Toto je prakticky realizování odchytáváním assertion vyjímek. U těchto testů je také zaručeno generování hlášení a perzistentní uchování výsledků.
Testovací systém byl inspirován metodikami testovacích rámců založených na programovacím jazyku Python. Inspirace pro automatickou indexaci a spouštění všech dostupných testů v rozhraní byla převzata z rámců Behave[2] a PyTest[3]. **[[blizsi popis link o dekoratorech?]]** V praxi je poté rozhraní definované složkou, ze které testovací systém načítá dostupné testovací sady. Testovací sada je blíže specifikovaná jako skript v jazyce Python, který obsahuje jednotlivé testovací případy. V systému se poté používájí anglické termíny suite, tedy sada, a test case, tedy testovací případ. Na obrázku níže je znázorněna struktura zdrojů testů pro takto definované rozhraní.



Obrázek 4.2: Příklad struktury složky nutné pro využití navrženého rozhraní. Názvy souborů mohou mít jakoukoliv formu, názvy v obrázku jsou strukturované pouze pro přehlednost. Důležité je využívat značkování testů pomocí dekorátorů.

[2]Informace dostupné z https://behave.readthedocs.io/en/latest/
[3]Informace dostupné z https://behave.readthedocs.io/en/latest/

Testovací případy jsou z pohledu rozhraní definované jako funkce. V Pythonu je funkce objekt, ostatně jako všechno jiné. To nám umožňuje objektu funkce přidávat atributy. To se používá u obecných testů (třídy 2 a 3) ke klasifikaci do příslušné třídy. Pro přiřazení atributů jsou použity vlastní dekorátory, které jazyk Python podporuje. Velmi podobnou mechaniku používá testovací nástroj Behave, odkud jsem se inspiroval.

4.5.2 Generalizace společných kroků testovacích případů

Testovací případ se z definice skládá z následujících kroků:

- Setup: Příprava prostředí pro vykonání testu
- Exercise: Provedení operací daného testovacího případu
- Verify: Vyhodnocení výsledků operací dle specifikací testu
- Teardown: Uvedení prostředí do původního stavu

V navrženém rozhraní se kroky *Setup* a *Teardown* v případě testovacích případů třídy 2 přesouvají dovnitř testovacího systému. Krok *Setup* je z pohledu testovacích případů třídy 2 převod RAW souboru. Samotný převod RAW souboru je ale také zaznamenáván jako testovací případ třídy 4. Tato generalizace je z hlediska návrhu velice důležitá, jelikož pokud by bylo potřeba převádět RAW soubor zvlášť pro každý testovací případ třídy 2, zabralo by to ohromné množství času a testování by bylo velice neefektivní. Jeden RAW soubor by tak musel být pro tři testovací případy v rámci jednoho běhu třikrát konvertován. Místo toho se vždy provede převod, jeho výsledek se zaznamená jako testovací případ třídy 4 a výsledek převodu, tedy soubor v obrazovém formátu, je poté přes rozhraní opakovaně poskytnut všem testovacím případům třídy 2. Stejný mechanismus se používá i pro integrované testy třídy 1. Stačí tedy provést krok *Setup* pouze jednou pro spuštění všech testů, jenž vyžadují na vstupu soubor. Tato mechanika je pro lepší pochopení popsána diagramem v obrázku B.2. v příloze B. Jelikož se v tomto projektu jedná o vstupem řízené testování a vstupní množina je velmi rozsáhlá, je tento proces nezbytně nutný. Generalizace společných procesů tříd testů byla zavedena až ve druhém sprintu vývoje. Prototyp systému v prvním sprintu se zaměřoval spíše na architekturu systému a používal se jen jeden pracovní testovací případ. Jelikož byl testovací případ jen jeden, nebyla tato generalizace potřeba. Při přidání dalších testů byla tato chyba v návrhu samozřejmě zjištěna a bylo potřeba značně přepracovat strukturu testovacího systému.

4.6 Návrh zapojení externích testů

Jak bylo avizováno v předchozích sekcích, pro produkt Zoner Photo Studio X existují základní jednotkové testy pro používané interní knihovny. Integrace těchto testů do systému by mohla být realizovaná pomocí spuštění testů v jejich standardním prostředí, tedy C++. Zaznamenáván by byl pouze binárního výsledek, tedy zda test prošel či nikoliv. Tento výsledek by se tak dal zachytit ze skriptu testovací sady korespondující k externímu testu, jenž by byl zapojen přes rozhraní do testovacího systému. Prakticky by pak každý testovací případ pouze spustil daný externí test, například přes volání operačního systému, a zaznamenal jeho výsledek. Výsledek by se pak již standardně uložil přes rozharní do testovacího

systému a byl by zapojen do hlášení. Tato část ale ještě vyžaduje další průzkum možností, implementaci a případnou optimalizaci tohoto návrhu. Slibně vypadá například knihovna *Boost.Python* [4], která poskytuje rozhraní mezi programy v jazyce Python a C++. Je to tedy plán spíše do budoucna tohoto projektu a v aktuální verzi testovacího systému nejsou externí testy zahrnuty.

4.7 Vstupní datová sada pro testování

Jelikož je tato práce zaměřena na vysokoúrovňové, vstupem řízené testování, je potřeba korektně enumerovat vstupní množinu testovaného programu. Jelikož je specifikace v tomto ohledu detailní, vstupní množina dat se dá relativně dobře popsat. Pro modul zpracování RAW souborů v Zoner Photo Studiu X je veden seznam podporovaných fotoaparátů. Vstupní množina se tak teoreticky skládá z kombinace podporovaných RAW souborů a jejich parametrů, které mohou mít na převod vliv. Tyto parametry pevně souvisí se specifikací RAW formátu popsaného v kapitole 3. Jedná se o následující parametry RAW snímku:

- Expoziční údaje
  - Hodnota ISO
  - Mód expozice - manuální, automatický, priorita clony
- Barevný prostor
- Teplota bílé barvy ve snímku
- Poměr stran snímku
- Orientace snímku
- Velikost RAW souboru - většina fotoaparátů podporuje možnost nastavit velikost RAW soubrů, často se jedná o možnosti large, medium, small. RAW soubory se pak liší i úrovní detailu.
- Komprese
- Bitová hloubka - počet bitů na kanál RGB modelu. Standard pro RGB je 8 bitů na kanál, některé RAWy používají až 16 bitů na barevný kanál.

Výsledná vstupní množina je pak kombinace modelu fotoaparátu, na který byl RAW snímek pořízen, a jeho parametrů. Pro každý podporovaný fotoaparát je tedy třeba spouštět všechny testy nad snímky s různou kombinací parametrů. Jelikož je tato množina ve výsledku pro všech 77 podporovaných modelů fotoaparátů enormní, bylo ji potřeba určitým způsobem zoptimalizovat a nalézt ekvivalentní třídy v dostupné datové sadě. Optimalizace vstupů je popsána v sekci 4.7.3. Velkou výzvou je také tyto data získat, jelikož volně neexistuje žádná strukturovaná databáze. Navíc je potřeba nashromáždit RAW soubory se všemi potřebnými parametry zmíněnými výše v seznamu. Akvizice těchto dat je nedílnou součástí této práce. Strukturovaná báze dat byla vytvořena v rámci projektu pro tvorbu vlastních DCP profilů Zoner standard. Tvorba DCP profilů je relativně komplexní záležitost a vyžaduje testování fotoaparátu v laboratorních podmínkách. Bližší popis je dostupný v sekci

[4] Informace dostupné z https://www.boost.org/doc/libs/1_80_0/libs/python/doc/html/index.html

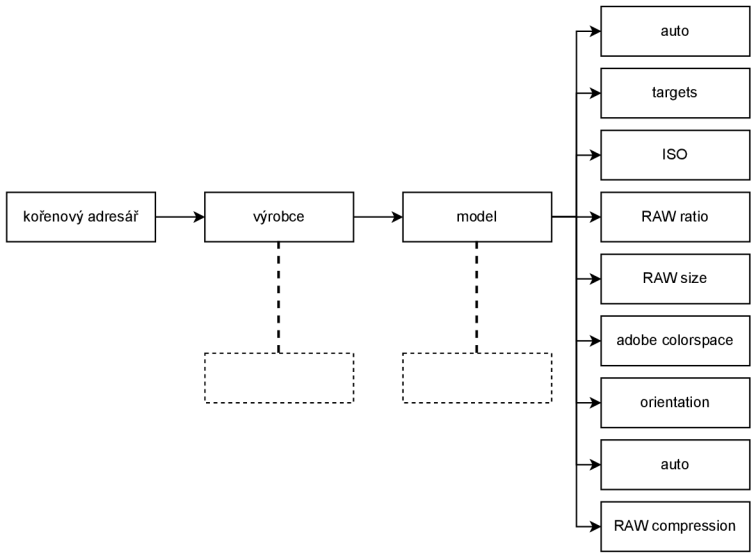
3.1.2. Tohoto projektu jsem se účastnil a pracoval jsem právě na tvorbě databanky RAW souborů, ze které mimochodem vychází i interní podpora vyprofilovaných fotoaparátů. Tato databanka se skvěle hodí pro účely testování a zaručuje přístup k RAW souborům i u nově podporovaných fotoaparátů. Mimo vzorky potřebné pro tvorbu DCP profilu obsahuje databanka i vzorky pro parametry popsané výše.

4.7.1 Nashromáždění datové sady

Samotné nashromáždění velkého množství RAW souborů je časově náročná práce hlavně kvůli nedostupnosti fotoaparátů. Toto bylo částečně řešeno přímou spoluprací s výrobci, kteří pro tento projekt poskytli fotoaparáty do zápůjčky. Komunikace s výrobci a domluvení zápůjčky bylo ale docela složité a často byl problém s domluvením termínů. Východisko nabídla spolupráce s lokální společností Fotori, jež fotoaparáty pronajímá. Většina vzorků RAW souborů byla tedy pořízena díky této oboustranné spolupráci.

4.7.2 Struktura datové sady

Po vytvoření vzorků jsou RAW soubory uloženy na serveru ve standardizované struktuře složek dle obrázku 4.3 níže. Jednotlivé složky jsou použity při testování kombinace parametrů. Nejpočetnější složky bývají typicky složky *targets*, *auto* a *ISO*. Na tyto je později aplikována optimalizace, jelikož jsou díky jejich parametrům snadno rozřaditelné do *ekvivalenčních tříd*. Navíc se čas testovacího běhu bez optimalizace vstupů pohyboval okolo 40 hodin a to ještě nad nekompletní vstupní množinou.



Obrázek 4.3: Struktura složek použitá pro ukládání RAW souborů. Listové složky obsahující RAW soubory se mohou lišit mezi jednotlivými modely, protože některé fotoaparáty například nepodporují možnosti komprese RAW souborů.

4.7.3 Optimalizace vstupní množiny pomocí ekvivalenčních tříd

Cílem tvorby ekvivalenčních tříd je při black box testování redukce vstupů při zachování co nejlepšího pokrytí kombinací vstupních parametrů programu, v ideálním případě pak pokrytí stejného. Metodika při vytváření ekvivalenčních tříd je taková, že se snažíme najít skupiny vstupů, které mají stejný nebo alespoň velice podobný vliv na průběh algoritmu a tím i na výstup testu. Poté stačí z této třídy vybrat jeden vstup, jenž celou třídu zastupuje, a ten použít při testování, jelikož se při správném rozdělení do tříd očekává, že test pro všechny vstupy z dané ekvivalenční třídy dopadne stejně.[10] Tento postup je použit při návrhu optimalizačního algoritmu pro automatické vybírání zástupců ekvivalenčních tříd v modulu `mapper`.
Jistá práce při tvorbě těchto tříd je předem hotová už díky vhodné struktuře v bázi vstupních dat, jelikož máme soubory relativně dobře uspořádané podle parametrů do složek.
Ekvivalenční třídy jsou tvořeny pouze pro RAW soubory ve složkách s názvy *targets*, *auto* a *ISO*, jelikož tyto složky obsahují většinou 80% všech vzorků RAW souborů per model. Jednotlivé **soubory** v ostatních složkách jsou již považovány za **ekvivalenční třídy samy o sobě**, jelikož jejich složka zahrnuje vstupní soubory přesně pro jeden typ parametru ze sekce 4.7, například *velikost RAW souboru*. Navíc je v tomto případě lepší netvořit ekvivalenční třídy, protože obsažené soubory se mohou dramaticky lišit napříč modely fotoaparátů a automatické vybírání zástupců ekvivalentních tříd by mohlo mít za následek vynechání celé ekvivalenční třídy.
Dělení souborů na ekvivalenční třídy ze tří výše zmíněných složek je provedeno takto:.

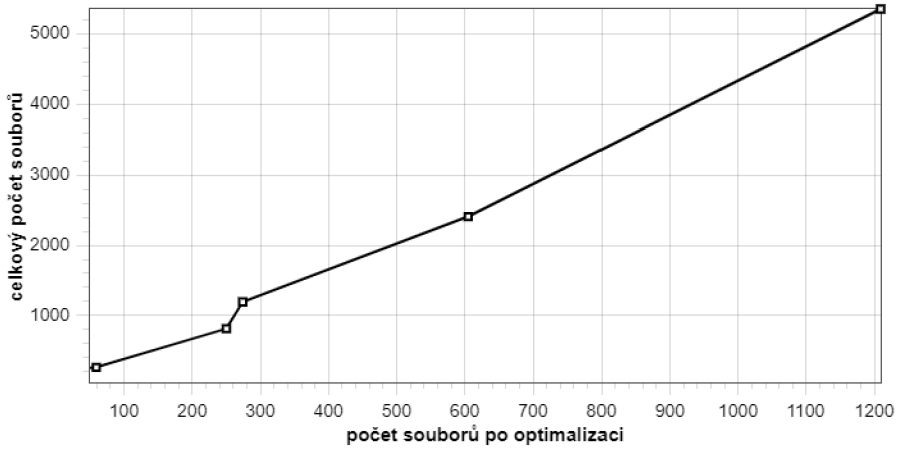
- **Složka targets** obsahuje vždy čtyři sekvence po devíti fotografiích. Tyto sekvence mají vždy stejné nastavení, ale jednotlivé fotografie v sekvencích se liší. Je tak odsud vybráno pouze prvních devět fotografií, protože ostatní již náleží do ekvivalenčních tříd definovaných těmito devíti

- **Složka auto** obsahuje snímky s automatickým nastavením. Všechny fotky tak lze řadit do jedné třídy dle parametru *mód expozice*4.7.

- **Složka ISO** obsahuje stejné soubory, pouze s rozlišnou ISO hodnotou. Jelikož ekvivalentní třída různých ISO hodnot je zastoupená prakticky v každém doposud vybraném vstupu, použijeme zde analýzu hraničních hodnot, která je obvykle velmi efektivní v odhalování chyb. Tato složka se pro to přímo nabízí, jelikož většina fotoaparátů používá dvě formy ISO hodnot: Nativní, těch je dosahováno hardwareově pomocí nastavení senzitivity senzoru, a rozšířené, ty jsou dopočítávány softwarově pomocí hodnot ze souboru RAW. Tyto rozšířené hodnoty ISO jsou vždy ty úplně nejnižší a nejvyšší z celkového rozsahu. Ve složce ISO jsou soubory vždy seřazeny podle hodnot ISO, vybereme odsud tedy první a poslední soubor, respektive soubor s nejnižší a nejvyšší ISO hodnotou, které jsou typicky dopočítávány v rozšířeném módu.

Algoritmizace tohoto vybírání zástupců ekvivalenčních tříd je popsána v sekci 5.2. Efektivita optimalizace hodně záleží na přesném počtu RAW souborů ve složce. Některé modely fotoaparátů mohou mít například velice málo souborů mimo optimalizované složky. Optimalizace se tak u nich zdá být efektivnější. Tak je to například v prvním záznamu listu `optimal` níže. Tato skutečnost je však spíše způsobena řazením fotoaparátu z tohoto měření mezi nižší cenovou třídu, která často některé vstupní parametry datové sady vůbec nepoužívá. Experimentální testování efektivity této redukce vstupní množiny je zobrazeno v grafu 4.4. Použité hodnoty vycházející z měření optimalizačního algoritmu. Lze si povšimnout, že až na vyjímku u počátku souřadnic (dvojice `o = 274, f = 1194`) je křivka lineární, což nám zaručuje, že při velmi vysokém množství vstupních souborů bude tento algoritmus stále stejně efektivní.

```
optimal: [  28,  59, 250,  274,  605, 1209 ]
full:    [ 240, 263, 813, 1194, 2410, 5355 ]
```



Obrázek 4.4: Graf efektivity optimalizace v závislosti na celkovém počtu RAW souborů ve strukturovaném adresáři.

4.8 Návrh systému pro obsluhu automatického testování Zoner Photo Studia X

Tato sekce pojednává o technických detailech, díky kterým byl realizován návrh popsaný v předešlé kapitole.

4.8.1 Vybrané technologie pro implementaci testovacího systému

Jako hlavní a prakticky jediný programovací jazyk jsem zvolil Python. Hlavním důvodem vedoucím k tomuto rozhodní je fakt, že Python má v důsledku své oblíbenosti nepřeberné množství výukových zdrojů na internetu. Je tak pro mě jednoduché vyhledat řešení problému, pokud bych se někde při programování zasekl. Další nespornou výhodou je jeho jednoduchost v základu a možnost doinstalovat pokročilé knihovny pro dosažení komplexnějších úloh. Zároveň Python plně podporuje modulární programování, které jsem zde využil. Modul je z pohledu Pythonu zdrojový soubor.

Balík je poté složka obsahující soubor `__init__` a jednotlivé moduly. Hlavní nevýhodou Pythonu je jeho nízká rychlost při vykonávání výpočetně náročných operací, jelikož se jedná o interpretovaný jazyk. V této práci však výkon nehraje naprosto kritickou roli a je to vykoupeno akcelerací vývoje díky jednoduchosti. Problém rychlosti je často v praxi řešen vytvořením rozhraní mezi Pythonem a C++. C++ v pozadí pracuje na obsluze výpočetně náročné operace a rozhraní pro tuto knihovnu je implementováno v Pythonu a lze jej tak v Python projektech velice snadno využít.
Pro perzistentní ukládání dat jsem zvolil databázový systém MongoDB[5] izolovaně spuštěný v Docker kontejneru[6]. Do systému byl dále integrován nástroj Allure pro vizualizaci a analýzu výsledků testů. Pro vytvoření síťového koncového bodu pro možnost spouštění testů automaticky a vzdáleně přes síť jsem zvolil webový framework *Flask*[7] a WSGI *waitress*[8].

4.8.2 Architektura systému

Již v návrhu v sekci 4.4 bylo zmíněno, že se bude pracovat s modulárním programováním a velice základním OOP[9]. Detailní dokumentace ve formě diagramu je k nalezení v příloze B. Pro implementaci využívám vždy jeden Python modul na jednu třídu. Modul typicky obsahuje stejnojmennou třídu, ze které se poté při běhu tvoří singleton objekt. Singleton je návrhový vzor, dle kterého se v systému vyskytuje vždy maximálně jedna instance třídy v jakýkoliv daný moment a je ze všech bodů programu globálně dostupná. Všechny navržené moduly, respektive třídy, jsou prakticky singleton, kromě třídy `logger`. Instanci této třídy si každý singleton objekt inicializujesám, aby s pomocí něj mohl zjednodušeně provádět výpisy do logovacích souborů, popřípadě na standardní výstup. Jeden z hlavních důvodů pro využití této architektury je možnost uchovávat stavy objektů, dle nichž hlavní řídící objekt poté provádí potřebné kroky k obsluze systému. V implementaci se nachází následující třídy, respektive singleton objekty. Předpona `ZT` znamená `ZTest`, což je pracovní označení navrženého testovacího systému. Balík obsahující moduly se zdrojovými kódy níže popsaných objektů je standardně nazván `src`.

- `ZTDriver` - hlavní řídící objekt, který vykonává veškerou operační logiku systému a inicializuje všechny další potřebné objekty, na něž drží reference po celý běh programu. Také spravuje rozhraní a externí testy.
- `ZTMapper` - objekt zodpovědný za přípravu vstupních testovacích dat.
- `ZTAnalyzer` - hlavní objekt pro ukládání výsledků testů a generování hlášení.
- `ZTRawConverter` - objekt zastřešující operace pro převody RAW souborů.
- `ZTDatabase` - objekt řídící databázové transakce nad MongoDB driverem.
- `ZTLogger` - objekt pro usnadnění vypisování stavových informací.
- `ZTBenchmarkChecker` - objekt zodpovídající za tvorbu benchmark souborů a kontrolu shody obrazových dat. Figuruje v testech třídy 1.

[5]Dokumentace dostupná z https://www.mongodb.com/docs/
[6]Docker je systém poskytující virtualizovné prostředí pro izolovaný běh aplikace. Dokumentace dostupná z https://docs.docker.com/
[7]Dokumentace dostupná z https://flask.palletsprojects.com/en/2.3.x/
[8]Dokumentace dostupná z https://docs.pylonsproject.org/projects/waitress/en/stable/index.html
[9]Oběktově orientované programování

Dále je v systému přítomný balík `endpoint`. Tento balík obsahuje moduly, které slouží ke spouštění systému. Konkrétně se jedná o moduly:

- `endpoint` - Flask aplikace poskytující koncový bod pro vzdálení spouštění testů přes síť. V momentální verzi není podporovaná, jelikož se změnil nárok na spouštění testování.
- `handle_request` - modul pro zpracování příchozího HTTP požadavku.
- `launcher` - modul poskytující zpracování argumentů z příkazové řádky a přímé spouštění testovacího systému pomocí instanciace `driver` objektu a volání jeho příslušné vstupní metody.

Poslední balík zdrojových souborů je nazvaný `utils`. Tento balík obsahuje modul `ztest` a `img_cmp`. Modul `ztest` poskytuje sadu dekorátorů pro zapojování externích testů do rozhraní. Druhý modul, `img_cmp`, poskytuje pouze funkci `compare()`, která slouží ke zjišťování obrazových rozdílů dvou souborů v pipeline pro integrované regresní testy.
Dokumentace tohoto návrhu je v podobě diagramů uvedena v příloze B. Diagram popisující postup při inicializaci těchto objektů v systému je vizualizován na obrázku B.1. Diagram celkového návrhu systému a toku dat lze nalézt ve stejné příloze, obrázek B.3.
Průběh testování na kódové úrovni a algoritmy jsou popsány v sekci 5

4.8.3 Způsob perzistentního ukládání výsledků pro analýzy v pozdější části projektu

Na začátku prvního sprintu byla navrhnuta pro ukládání dat SQLite databáze. Ta se ale ukázala být nevhodná, jelikož je potřeba počítat s nekonzistentním formátem výsledků testů na rozhraní. Výsledky testů musí být příjimány v takovém formátu, jak uzná za vhodné návrhář tohoto testu, aby byla zachována dostatečná generalizace v rozhraní. NoSQL databáze MongoDB, implementovaná ve druhém sprintu, se ukázala být vhodnou, jelikož neukládá data v předem definované struktuře a je tak možné volně definovat záznamy typu klíč-hodnota. Nad těmito daty pak může být volně navržena pokročilejší analýza. Tvrzení o nevhodnosti SQL databáze lze oponovat, jelikož záznamy typu klíč hodnota lze serializovat a ukládat do velkých textových polí. V tomto přístupu by ale v budoucím vývoji mohla eskalovat složitost. U NoSQL databáze je její předností jednoduchost.

4.8.4 Běhové prostředí testovacího systému, jeho integrace a nutné prekvizity

Testovací systém potřebuje relativně velké množství externích programů a podmínek, aby mohl testovací cyklus proběhnout. Bylo tedy navrženo, že v rámci firmy bude vytvořena virtuální stanice se standardním systémem Windows, který je nutný pro běh testovaných komponent Zoner Photo Studia X. Do této virtuální stanice je testovací systém plně integrován. Prekvizity pro spolehlivý výkon testování jsou následující:

- Allure 2.21 - tento rámec navíc vyžaduje instalaci prostředí Java.
- Python 3.10 nebo vyšší. Interpret Pythonu je v nasazení na pracovní stanici izolován ve virtuálním prostředí.
- Veškeré python balíčky popsané v příloze `requirements.txt` - C.4

- Správné nastavení cest v konfiguračním souboru `config.yml`C.3
- Instalační programy pro vyžadované produkční kanály. Pro automatické spouštění je navíc potřeba upravit proměnné prostředí systému Windows, aby testovací systém odchytil zprávy o instalaci nové verze.
- V souvislosti s předchozím bodem je pro spuštění testů nutné mít nainstalované Zoner Photo Studio X.
- Docker Desktop pro Windows s MongoDB image. Kontejner musí být spuštěný.
- Přístup k databázi zdrojových souborů přes interní síť firmy Zoner, popřípadě přes správně nakonfigurovanou VPN. Databáze musí být korektně nastrukturována dle popisu na obrázku 4.3

Pro zasazení do kontextu v postupu projektu jsou momentálně všechny výše uvedené podmínky splněny. Testovací systém ale není v plném provozu. Okolnosti jsou popsány v kapitole 6

4.8.5 Automatické spouštění systému

Dle návrhu má mít automatické spouštění dvě možnosti. Obě jsou vyobrazené v příloze B.3. V praxi se ale spouštění přes endpoint ukázalo být příliš složitým, jelikož by se musely složitě instalovat aktualizace Zoner Photo Studia X na virtuální server. Místo toho se využívá integrace testovacího systému přímo s instalátorem Zoner Photo Studia X. Pokud je povolen vývojářský režim pomocí souborů pro úpravu registru systému Windows, instalátor po dokončení práce na aktualizacích provede jednoduché systémové volání testovacího systému, ve kterém předá mimo statických argumentů také proměnné argumenty, jako je verze, typ produktu a update. Pokud je volání provedeno správně, testovací systém se spustí a vykoná testovací cyklus definovaný tímto voláním. Na konci testování pak systém vygeneruje hlášení o výsledcích testu.

4.8.6 Hlášení výsledků testování

Pro hlášení výsledků testování a jejich přehlednou prezentaci byl do testovacího systému integrován otevřený reportovací rámec *Allure*.[10] Ten je sice primárně určen pro integraci s jinými testovacími nástroji, ale v rámci experimentů se mi podařilo najít způsob, jak výsledky testů přávadět na strukturovaný JSON[11] soubor, který Allure přijímá. Stačí vygenerovat strukturu v určitém tvaru. K tomuto se mi bohužel nepodařilo najít dokumentaci, ale při experimentování s frameworkem PyTest se mi podařil tento formát nastudovat a zreplikovat jeho generování. Příklad výsledku testu připraveného pro export do Allure projektu je dostupný v příloze D. Report také zobrazuje veškeré informace o výsledcích i době trvání testu. Při opakovaném spouštění testů se pak v Allure projektu začne tvořit i historie a docela pěkné statistické grafy. V příloze jsou k nalezení snímky obrazovky hlavního panelu pod označením D.1, a také demonstrace zobrazení historie testovacího případu, obrázek D.2. Dále jsou také přímo do reportu přidány informace o verzi testovaného Zoner Photo Studia X v souboru `environment.properties`.Veškeré tyto operace vykonává objekt `analyzer`, který tato data dostává přímo po dokončení testu od řídícího objektu `driver`. Příklad je zobrazen v příloze D. V průběhu testování jsou průběžně tisknuty na standardní výstup

[10] Dokumentace dostupná z https://docs.qameta.io/allure/
[11] Java Script Object Notation

pomocí objektu `logger` výsledky skládající se z identifikátoru testu a z použitého vstupního RAW souboru. Například

`PASS suite_size#ejpg_size with C:<cesta>\Nikon\Nikon Z fc\auto\DSC_0305.NEF`

pro úspěch nebo

`FAIL suite_size#ejpg_size with C:<cesta>\Nikon\Nikon Z fc\auto\DSC_0305.NEF`

pro selhání testu. Popřípadě modul `logger` také vypisuje následující hlášku, pokud je test z nějakého důvodu přeskočen:

`WARNING suite_size#ejpg_size with C:<cesta>\Nikon\Nikon Z` Forma `<nazev_sady>#<nazev\test` je používána všude v systému pro jednoznačnou identifikaci testovacího případu. Používá se také jako názvy v Allure reportu.

4.9 Testovací sada pro rozměry souborů

V rámci požadavků firmy byly také jednoduché testy na porovnání rozměrů souborů převedených z RAW souboru za pomocí modulu Zoner Photo Studa, AdobeDBG a také pro zabudovaný náhledový JPG soubor z fotoaparátu. Výsledné rozměry by měly být v modulu ZPS X stejné jako u Adobe i zabudovaného JPG souboru. Testy jsou navrženy jako prototypy pro validaci návrhu rozhraní pro přidávání dalších test. Zdá se, že je tento přístup ideální, jelikož implementace těchto testů byla otázka několika minut díky automatickému překládání RAWů a automatické indexaci a spouštění těchto testů. Příklad kódu použitého pro implementaci testů pro toto srovnání velikostí jsou k nalezení v sekci **??**

4.10 Návrh regresního a výkonnostního testování převodníku RAW souborů náhled do aktuálního stavu

Specifikace tohoto testu je již relativně jasně vymezená z předchozích kapitol. S každým vydáním nové verze Zoner Photo Studia X je třeba ověřit, zda nedošlo k neočekávaným změnám obrazových dat nebo výraznému zpomalení procesu při převodu jakéhokoliv podporovaného RAW souboru.

4.10.1 Regresní kontrola obrazových dat, její návrh a postup v implementaci

Regresní kontrola obrazových dat je navržena podle principu tvorby benchmarkovacích souborů, které budou manuálně prozkoumány a prohlášeny jako dobrá výchozí hodnota pro budoucí ověřování obrazových dat v nových verzích Zoner Photo Studia X. Z hlediska realizace je toto také nejspíše nejefektivnější přístup, ač vyžaduje relativně velké vstupní usilí pro ustanovení první generace výchozích souborů. Obměna těchto souborů v důsledku zvýšení kvality například s vylepšeními v oblasti konverzního modulu pak již nebude náročná a celý tento systém působí dosti stabilně. Samotný návrh implementace je velice jednoduchý. V aktuální stabilní verzi Zoner Photo Studia X se nechá převést kompletní databanka RAW souborů, optimalizovaná dříve navrženým dělením do ekvivalenčních tříd, do souborů TIFF. Soubor TIFF je vybrán zejména díky jeho bezztrátové kompresi. Tyto benchmark soubory budou umístěny ve stejné složce, jako je jejich originální RAW podoba. Struktura

databanky zůstane zachována, a benchmarkovací soubory si zároveň ponechají jméno původního souboru, ze kterého byly převedeny, rozšířené o příponu `*-BENCHMARK` a změněnou příponu `*.TIFF` v důsledku konverze. Toto poskytne snadné párování orignálních RAW souborů k jejich benchmarkovacímu TIFF souboru. Porovnávání dvou obrázku bylo navrženo porovnávání přesných hodnot jednotlivých pixelů a jako hlavní metrika rozdílnosti dvou TIFF souborů byl zvolen počet lišících se pixelů, jelikož i malá změna v expozici bude mít vliv téměř na všechny pixely ve výsledném obrazovém formátu. Detailní postup této validace shodnosti je pospán v sekci **??** Tento návrh je již plně implementován do testovacího systému, ale již při interním testováním tohoto postupu byla objevena docela závažná chyba, která odsunula možnost tento navržený postup validovat. Interní testování bylo prováděno na malé sadě dat, ale žádný ze souborů nebyl identický. Detailní popis tohoto nálezu je popsán v kapitole **??**. Tato závada testovaného systému bohužel dočasně znemožnila tento postup plně implmentovat a zapojit do produkce.

4.10.2 Návrh kontroly poklesu výkonu v modulu pro konverze RAW souborů

Výkonnostní testování RAW převodníku spočívá hlavně v měření času od začátku konverze RAW souboru po jeho uložení na pevný disk počítače, respektive virtuální stanice na které je momentálně testovací systém nasazen. Další důležitý faktor bude zatížení operační paměti. Vzhledem k navržené architektuře testovacího systému dává smysl tyto úraje společně s unikátním identifikátorem RAW souboru ukládat do databáze. Opět se pro každý RAW soubor stanoví nějaký benchmark, který může být v tomto případě složen z průměru několika posledních běhů, nebo dokonce kontrolovat odchylku v nové verzi od celkové průměrné doby nutné pro převod daného RAW souboru. Tento postup však v aktuální verzi není ještě naimplementován ale jsou pro něj připravené podklady. V momentální verzi lze z databáze extrahovat data o proběhlých testovacích převodech RAWu, ale není zatím zhotovena analýza, kromě grafů rozložení potřebného času v Allure reportech. Jelikož je hlavní cíl, tedy kontrola obrazových dat, momentálně pozastaven kvůli nalezení neočekávané chyby, měl by tento testovací případ získat v začátku dalšího sprintu tohoto testovacího projektu prioritu, pokud bude výzkum zmíněné chyby blokovat úplné dokončení testů obrazové shody dlouho. Momentální plán je nejdříve ale unifikovat jeden převod pro všechny operace, což bude do dalšího sprintu vyžadovat lehké strukturální změny toku dat v testovacím systému. Mimo tento koncept byl také implementován generátor stresu, tedy zatížení procesoru pro experimenty s dobou trvání převodu v závislosti na umělém zatížení systému. Tato možnost je nasazená v aktuální verzi, ale pouze není zatím plně automatizovaná. Je možné systém zatižít pouze na určitý čas, není vhodé. Do budoucna by bylo nutné vytvořit generátor zátěže který by vyvíjel nátlak na systém pouze během běhu testovaného systému. Tomuto se ale spíše věnuje oblas stres testování.

4.10.3 Algoritmus pro porovnávání obrazových dat

Jak bylo nastíněno v 4.10.1, tato část testovacího projektu je nyní pozastavena a čeká se na opravení chyby. Detaily chyby jsou popsané v kapitole 6. Aktuální algoritmus pro kontrolu shodnosti dvou obrázků je velice jednoduchý ale efektivní. Postup algoritmu je následující:

- Jelikož algoritmus přímá na vstupu cestu ke dvěma obrázkům, je třeba oba nejdříve načíst do paměti. To je dosaženo použitím knihovny `imageio.v2`, konkrétně funkce `imread()`. Tanto funkce vrací `numpy` pole s načtenými obrazovými hodnotami.
- Jelikož používáme numpy pole, můžeme od sebe obě matice barev přímo odečíst. Výsledek je matice stejné velikosti, která obsahuje rozdíly barevných hodnot na jednotlivých pozicích.
- Nyní stačí spočítat počet pixel, které jsou nenulové. V matematice se této operaci občas říká L0 norma, ač se o normu nejedná. Pokud nám výsledek vyjde nula, tak víme že obrazy na vstupu jsou totožné a z pohledu testování je toto pro nás úspěch testu. Nevýhoda tohoto algoritmu je ta, že nedokáže porovnat dva snímky různí velikosti. To ale není v případě aplikace pro regresní testování problém, jelikož budeme vždy porovnát stejné snímky. Jediný rozdíl je v tom, že budou konvertovány v jiné verzi. Pokud by například tento algoritmus na nějaké snímku selhal, lze to považovat za podezřeníhodné na změněné rozměry.

Algoritmus ve zjednodušené formě bez odchytávání vyjímek vypadá následovně:

```
def compare(f1, f2)
    img1 = imread(f1).astype(float)
    img2 = imread(f2).astype(float)
    difference = img1 - img2
    zero_norm = norm(difference.ravel(), 0)
    return zero_norm
```

Tento zdrojový kód byl inspirván zdrojem z [4]

Kapitola 5

Implementační detaily testovacího systému pro Zoner Photo Studio X a souvislosti s návrhem

V této sekci jsou vybrány některé tématiky z implemetace navrženého testovacího systému a jejich návrh je probrán na úrovni kódu. Velice doporučuji před jejím čtenám projít dokumentační diagramy (příloha B nebo kód a komentáře v něm. To pomůže při pochopení architektury lépe než text. Argumenty spouštění programu jsou též dostupné v příloze **??**

Pro upřesnění pojmů, benchmark je v kódu pracovní název pro modul určený k provádění operací v rámci regresního testování obrazové shody.

5.1 Aktuální možnosti spouštění testovacího systému

Jak již bylo sepsáno v sekci o modulárním návrhu v sekci 4.8.2 výše, systém je rozdělený na několik modulů a každý objekt z korespondujícího modulu zastává svojou určitou funkci. Uprostřed stojí `driver`, který veškeré procesy orchestruje. Momentálně existují tři možnosti spuštění testovacího systému. *Spuštění všeobecných testů*, *spuštění regresních testů pro porovnání obrazové shody* a *generování nových benchmarkovacích souborů pro porovnání obrazu*, podle kterých se pak porovnává obrazová shoda v nových verzích programu. Toto řešení je poněkdu neštěstné a v budoucích fázích tohoto projektu se chci zaměřit na kompletní unifikaci překladu RAWu, jelikož momentálně nelze spustit včechny operace zároveň. Všechny vstupní argumenty jsou dostupní v příloze C.1.

5.1.1 Nastínění průchodu testovacím cyklem na úrovni objektů

Následující část je slovní popis obecného běhu programu. Důrazně doporučuji při jeho čtení nahlížet na diagram B.3.

Všechny tři běhy ze sekce výše mají určitou společnou část, kterou je pro `driver` inicializace ostatních modulů, pro `mapper` načítání vstupů a například pro `database` je to připojení k MongoDB serveru. První vstup do programu je však buď od uživatele manuálním spuštěním skriptů, z Flask koncového bodu po obdržení HTTP požadavku nebo spuštěním skriptu automaticky po instalaci nové verze Zoner Photo Studia X. Pokaždé se

ale první spustí launcher který zpracuje argumenty a vytvoří řídící objekt `driver` Po výše zmíněné inicializaci jsou tedy tři možnosti, všechny jsou volání metod objektu driver:

1. `run_benchmark_generation()` spustí generování nových benchmarkovacích dat. Hlavní roli zde hraje objekt `benchmark`, který pro všechny vstupy, případně optimalizované objektem `mapper` postupně v cyklu volá `converter`, který generuje TIFF soubory.

2. `run_testing()` spouští testování převodu zabudovaného testu pro převod RAW souborů s měřením času. Zároveň jsou převedené převedené obrazové formáty cyklicky poskytnuty všem testům na rozhraní, jenž nesou dekorátor `image_test`. Poté jsou ještě spuštěny všechny testy na rozhraní bez potřeby obrazových dat na vstupu.

3. `run_benchmark_testing()` spouští regresní kontrolu obrazových dat. Objekt `driver` v tomto případě opakovaně `benchmarker`, který pomocí utility `imgcmp` postupně provádí testy na stoprocentní obrazovou shodu.

Na konec běhu jsou doukládány všechny záznamy do databáze a pokud je specifikovaný přepínač `-r`, tak se spustí Allure report. Všechny argumenty pro suštění jsou dostupné v příloze C.1.

5.2 Detaily algoritmu pro optimalizaci vstupní množiny

Algortimus modulu `mapper` pro selekci optimálního vstupního setu není nijak složitý. Využíva předem definované indexy v konfiguračním souboru, aby se dal případně snadno přenastavit při změně požadavků na ekvivalenční třídy. Samotný algoritmus se skládá z následujících kroků:

1. Nejdříve se vytvoří Python dictionary, respektice slovník, říkejme mu `result\_dict`.

2. `result_dict = {}`

3. Algoritmus nejdříve načte do `result_dict` všechny definované optimalizační názvy složek, například ISO a auto, ve formě klíče. navíc se přidá obecný klíč `others`

4. `result_dict = {auto: "", iso: "", others: ""}`

5. Jako hodnota pro ty klíče se vytvoří další slovník.

6. `result_dict = {auto: {}, iso: {}, others: {}}`

7. Poté prochází jednotlivé složky kamer

8. V každé složce dané kamery se načte seznam existujících podsložek a ten se prochází.

9. Pokud podsložka obsahuje v názvu nějaký existující klíč slovníku `result_dict`, je složka přidána jako klíč do slovníku, jenž je hodnotou tohoto klíče v `result_dict`. Jako hodnota pro složku se vytvoří seznam.

10. `result_dict = {auto: {canon/auto: []}, iso: {}, others: {}}`

11. Pokud ne, je taková složky přidána jako klíč podslovníku představujícího hodnotu obecného klíče - `others`.

12. `result_dict = {auto: {canon/auto: []}, iso: {}, others: {ab/cd: []}}`

13. Nyní se seznam složek naplní soubory dostupnými v dané složce.
14. `result_dict = {auto: {canon/auto: [XYZ.RAW, ABC.RAW]}, ... }`
15. Nyní se indexy z konfiguračního souboru přemapují do seznamu každé složky, podle nejvyšších klíčů v `result_dict`. Tyto soubory se vyjmou ze seznamu složky a přemístí do finálního seznamu optimalizovaných souborů.
16. pro příklad níže berme v potaz, že index pro klíč auto je `0`. Vybere se tak soubor `XYZ.RAW`, název souboru se spojí se složkou a celá cesta se přesune do výsledného seznamu.
17. `result_dict = {auto: {canon/auto: [_______, ABC.RAW]}, ... }`

5.3 Implementační detaily rozhraní pro přidávání testů

Přidávání testů ze složky rozhraní je řešeno velice jednoduše pomocí dekorátorů. Dekorátor je sám o sobě funkce, jejíž vstup je jiná funkce a také tu samou funkci po určitých modifikacích vrací. V modulu `ztest`, jenž v rozhraní slouží jako poskytovatel funkcí testovacího systému, je například tento velice jednoduchý dekorátor:

```
def image_test(func):
    func.requires_file = True
    return func
```

Tento dekorátor pouze přidá objektu funkce atribut a je poté možné různé testy, představované těmito funkcemi, snadno odlišit a zařadit do adekvátní fronty k provedení.
Pro ověření je v modulu `ztest` vedena ověřovací funkce pro každý typ dekorátor:

```
def requires_file(func):
    try:
        if func.requires_file:
            return True
    except AttributeError:
        return False
```

Při používání dekorátorů stačí naimportovat soubor a použít řečený dekorátor funkce. Toto je příklad testu nasazeného v rozhraní systému s využitím dekorátoru.

```
from utils import ztest
@ztest.image_test
def adng_ratio(file_dict):
    zraw_width, zraw_height = Image.open(file_dict['zraw']).size
    adng_width, adng_height = Image.open(file_dict['adng']).size
    assert (zraw_width / zraw_height == adng_width / adng_height)
```

5.4 Implementační detaily zastřešení převodu RAW souborů

Modul `converter` nepřevádí RAW soubory pouze z RAW modulu ZPS X, ale využívá i CLI Adobe DNG a také extrahuje zabudované JPEG náhledy, pro případ, že by to nějaký test potřeboval. Toto využívá momentálně sada `suite_size`, které právě ověřují rozměry RAW souborů ze Zoner Photo Studia oproti převedeným souborům od Adobe a oproti zabudovanému JPEGu. Zároveň je toto ale zbytečné pokud `converter` bude převádět pouze RAW z modulu Zoner Photo Studia pro regresní obrazovou kontrolu. Jsou tedy vytvořeny tři funkce, které berou na vstupu cestu k RAW souboru. RAW je externě převeden voláním API Zoner Photo Studia a výsledek je uložen do dočasného úložiště. Jako výsledek tyto funkce vrací pro Adobe DNG a ZPS modul údaje o časovém trvání. Navíc je vytvořena funkce `convert_all`, která tyto informace vrací ve slovníku sestávajícího se z klíče v podobě převodního algoritmu (Zoner/Adobe/EJPEG) a čas. Toto je prováděno pouze v případě komunikace s objektem `driver`, který tyto data nechává uložit do databáze pro možnou regresivní kontrolu. navíc ale `converter` poskytuje i funkce pro `benchmark`, které berou na vstupu cestu k souboru a cestu k zápisu výsledku. Je tedy potřeba se nenechat zmást v tom, která funkce náleží komu. Přemýšlel jsem také nad implicitním jmenování funkcí, abych měl přehled v tom, který objekt volá jakou funkci. Příklad je pak třeba funkce `convert_zraw_for_benchmark()`. Myslím, že toto kódu dodává na přehlednosti.

5.5 Dokumentace kódu a clean code přístup

Všechny moduly jsou důkladně komentovány standardními dokumentovacími řetězci Python, které vychází z konvence PEP 8[1]. Blokové komentáře jsou v kódu přítomné u všech funkcí a tříd. Řádkové komentáře využívám v místech, kde považuji kód byť jen minimálně za nejednoznačný. Zároveň jsem při psaní kódu využíval automatické formátování, takže je vizuální styl kódu jednotny a dobře čitelný.

5.5.1 Napovídání datových typů

Python je jako takový dynamicky typovaný jazyk, existuje ale možnost přidávat takzvanné *type hints* ., které se dají přeložit jako *nápovědy k typům*. Jejich využití sice nemá ve výchozím nastavení vliv na standardní Python interpret, ale umožňují dosáhnout vyšší přehlednosti kódu. Typové nápovědy jsou také často využívané integrovanými vývojovými prostředími a pomáhají udržení typové korektnosti. Navíc výrazně ulehčují práci na větších projektech. Ve zdrojovém kódu tuto možnost využívám minimálně ve všech paramatrech funkcí, jelikož mi IDE PyCharm velmi ulehčuje práci, pokud má typovou nápovědu k dispozici. Zároveň je toto přínos k čistotě kódu a obecně se jedná o dobrou praktiku.

5.5.2 Vygenerovaná dokumentace

K dispozici je také dokumentace vygenerovaná nástrojem *pydoc* . Nepovažuji ji ale za příliš kvalitní a spíše pro pochopení kódu doporučuji využívat diagramy v příloze B a komentáře v kódu.

[1] Více informací je dostupných z https://peps.python.org/pep-0008/

Kapitola 6

Evaluace testovacího systému, jeho efektivity a navržené testovací sady

6.1 Celkové vyhodnocení postupu v testování produktu Zoner Photo Studio X

Od začátku projektu se podařilo vytvořit pro produk Zoner Photo Studio X prototyp ucelené integrované platformy, která je schopná automaticky detekovat, spouštět a vyhodnocovat testy, včetně řešení pro spolehlivou přehlednou prezentaci a uchování výsledků testů pro možnosti budoucích analýz. Povedlo se zautomatizovat proces spouštění a reportování testů. Také bylo zajištěno stabilní běhové prostředí na firemních serverech, takže se dá spolehnout na to, že testy které typicku budou běžet přes noc neskončí uprostřed a nezahodí veškerý progres. V konečném důsledku byla nalezena i jedna hodně zajímavá chyba popsaná blíže v sekci 6.5.
Původní očekávání se týkalo pouze funkčního programu pro efektivní regresní testování. Výsledný systém této specifikace dosahuje, jelikož spolehlivě rozpozná i okem neviditelné rozdíly. Zároveň byla testována i matematická metoda na počítání pixelů v umělém prostředí, například nad uměle vytvořenými soubory korektně vypočítá rozdílné pixely.[1]

6.2 Vyhodnocení Regresního testování obrazu

Regresní testování obrazové shody dosáhlo stanoveného úroveně efektivity, jaké bylo požadováno ze strany firmy. Bohužel ale nebyla možnost jej otestovat, protože celý systém byl dokončen relativně pozdě a ještě ve fázi testování byla objevena docela zásadní chyba v převodníku RAW souborů, kvůli níž nebylo možné tento návrh použít v produkci, jelikož chyba ještě není opravena. Na druhou stranu je cílem testovaní chyby odhalovat a to se v tomto případě podařilo, jen se tak při přípravách na ostré nasazení a vyhodnocí efektivity regresního testování. V demonstračních podmínkách tento systém fungoval dobře a byl pokaždé schopný jakoukoliv změnu detekovat. Dle mého uvážení by ovšem byla potřeba lepší metrika než je počet odlišných pixelů v obrázku. Lepší by bylo například generovat z proměnné `difference` z algoritmu v sekci 4.10.3 černý obraz, kde by se absolutní hodnota rozdílu mapovala do bílé barvy a byly by tak například vidět obrysy míst, na kterých se vstupní soubory liší.

[1] Větev dedikovaná těmto testům je dostupná na https://github.com/terrorgarten/ZTest/tree/imgcmp_tests/src

6.3 Vyhodnocení vykonnostního testování

Výkonnostní testování dosáhlo nižší úrovně než jsem čekal a nepodařilo se ho prozatím automatizovat. Nejlepší přehled o testování rychlosti převodů je dostupný v grafech Allure reportů. I odsud se ale dá vyhledat určité modely fotoaprátů pro které by mohly být převody optimalizovány, jako je například *Canon EOS 5D Mark IV.*, jehož převod trvá v průměru o sedm vteřin déle, než ostatní ostatní Canon zařízení. Předpokládám, že je to způsobené, ale v rámci reportu z dosavadního testování to s kolegy z firmy prověříme. Nicméně bych se rád této oblasti věnoval v příštím sprintu projektu.

6.4 Vyhodnocení automatické optimalizace datové sady pomocí ekvivalenčních tříd

Automatická optimalizace modulu `mapper` projektu velmi pomohla. Při spuštění testů nad celou datovou celou pracovní datovou sadou bylo dosaženo 77% časového zlepšení, což po nasazení do ostré produkce může znamenat ušetření přes 29 hodin strojového času na jedno spuštění. Navíc se i při spuštění na celé datové sadě, tedy na všech 77 fotoaparátech dostane dle propočtů čas lehce nad deset hodin. Dle zmínky v sekci 2.4.1 tak systém splní podmínku „běhu přes noc“, což byl od začátku jeden z hlavních cílů výzkumu ekvivalenčních tříd.

6.5 Nalezené nedostatky v testovaném systému

Během testování nad zkušební sadou byla nalezena jedna relativně závažná chyba při použití regresního testování. Z návrhu v sekci 2.4 vychází jako nejlepší způsob zaručení stoprocentní obrazové kvality použití porovnání per pixel, tedy srovnávání hodnot každého pixelu s jemu odpovídajícím ve druhém snímku.
Při interním testování jsem narazil na problém, kdy vůbecd žádné snímky nedávaly nulový výsledek z porovnávacího algoritmu. Co bylo o to zvláštnější, tak byl fakt, že i když jsem nechal konvertovat ten samý soubor vícekrát a výsledné TIFF soubory jsem pak mezi sebou kombinovaně porovnával, vycházely stále velice vysoké hodnoty rozdílu, například 9761303. Tato hodnota je skutečný výsledek porovnání dvou obázků vzniklých konverzí z jednoho samého. Po výzkumu a konzultaci s kolegou jsme vyzkoušeli použít vizualizaci na zvýraznění rozdílů takové dvojice obrázků a zjistili jsme že se opravdu každá dvojice převedená modeluem Zoner Photo Studia liší. Nikdy se žádné dva soubory nelišili stejně, ale určitý trend v opakovaném vyvolávání šlo pozorovat. Příklad zvýraznění rozdílu je v příloze E.1. Zjsitili jsme, že se jednalo o chybu v pipeline převodu RAWu. Konkrétně je problém nejspíše v sekci odšumu, ale chyba zatím není opravená, tudíž nemohu s přesností konstatovat, proč se toto dělo. Na tomto jevu bylo velice zvláštní, jak nedeterministický byl.
Další chyby byly nalezeny při standardním testovacím spouštění v sadě `suite_size`. Překvapivě, došlo 678 selhání z celkových 2144. Tuto situaci budu hlásit kolegům.

V důsledku těchto informací se domnívám, že pokud i základní testovací případy neprojdou, možná by firma měla do testování investovat více peněz.

6.6 Známé nedostatky návrhu a implementace

Testovací systém navržený v rámci této práce má problém v tom, že všechny testovací případy nejsou integrovány tak, aby bylo možné provést všechny testy nad jediným překladem

RAW soubor. Toto budu řešit restrukturalizací hlavního řídícího objektu, kde se pokusím vytvořit nějaký jednotný způsob, jak předat cestu k RAw souboru všem dostupným testovacím sadám a nevybírat mezi integrovanými a neintegrovanými, jak je tomu nyní.
Bohužel se také nepodařilo nasbírat dostatek dat pro vyhodnocení efektivity tohoto způsobu testování během tohoto sprintu, ale i tak systém odhalil byť triviální, ale doposud z nějakého důvodu všem skryté a důležité chyby.

6.7 Další plány v rozvoji projektu za rozsah této práce

Momentální plán v pokračování tohoto projektu je nejdříve opravit chybu blokující plné nasazení regresních obrazových testů. Dále je v plánu zaintegrovat i srovnávání výkonu mezi verzemi do automatické analýzy a strukturální změny `driver` modulu. V blízké budoucnosti je pro tento projekt také v plánu vytvořit rozsáhlou regresní testovací sadu na kontrolu korektnosti manipulace s metadaty, jelikož se proto problematiku zavádí nové zpracování a bude potřeba jej také testovat. Testovací sada pro tuto oblast bude nejspíše rozsáhlejší než ta aktuální, proto by bylo vhodné zvážit výzkum tvorby unifikovaného prostředí pro testy, pro které bude automaticky poskytnuta i regresní analýza výsledků.

Kapitola 7

Závěr

V bakalářské práci se mi podařilo vytvořit integrovaný systém pro automatické spouštění testů pro modul převodu snímků RAW v programu Zoner Photo Studio X. Tento testovací systém je navrhnut a vyvinut tak, aby obstál při konfrontaci s velice specifickým nárokem na testování, obzvláště pak pro fotografický software. Výjimečnost tohoto systému spočívá také v tom, že poskytuje otevřené rozhraní pro testy, které nutně potřebují na vstupu převedený RAW soubor, aby nad ním poté mohly kontrolovat různé údaje a validovat, že byl převeden korektně. Velkou výzvou také bylo celý převod RAW souborů zautomatizovat a využít jeden převod pro obsloužení co nejvíce testů, jelikož se v kontextu testování software jedná o dlouhý proces. Jeden RAW soubor může zabrat konvertoru až dvacet vteřin storjového času a pokud má tento testovací systém pokrýt všechny všechny přípustné vstupní RAW soubory, je potřeba vytvořit ekvivalenční třídy, aby se neplýtval čas čekáním na doběhnutí testů v agilním vývojovém prostředí firmy. Zároveň je ale potřeba nevynechat žádný unikátní vstup, jelikož konkurence v oblasti fotografického softwaru je dnes velice silná a zákazníci, tedy fotografové, nepřipouští žádné chyby, kterých se Váš software dopustí na jejich díle. Proto bylo potřeba s rozvahou vybírat vstupní množiny a najít dobrý bod, ve kterém dlouhé trvání testů nevykolejí vývojový proces a zároveň neohrozí kvalitu. Toho bylo docíleno pomocí pečlivého vytvoření ekvivalenčních tříd na základě samostudia a účasti na vylepšování stávajících metodik v této firmě. Ekvivalenční třídy jsou zároveň kombinovány s výběrem známých hraničních hodnot z množiny parametrů RAW souborů, které mají na převod vliv. Výsledkem je tedy 70% zrychlení testovacího cyklu s minimálními ztrátami na unikátních vstupech. Při vývoji testů byly použity poznatky z grey box testování, tedy hybridním způsobem, kdy jako tester nevidím do úplných detailů systému, ale jsem obeznámen s významy jednotlivých parametrů a jejich vlivem na možné chování systému. Tímto se dosáhlo regresní vizuální kontroly, která zaručuje, že se mezi jednotlivými verzemi nemění ani jeden pixel. Případně ale lze toleranci nastavit. Za hlavní přínos ale považuji to, že by ve firmě vzbuzen zájem o testování. Rozumím tomu, že pro většinu vývojářů není příjemné opouštět komforntí zónu a pouštět se do něčeho nového, ale jsem rád, že se to povedlo a že tento projekt bude pokračovat i za hranicemi bakalářské práce.

Literatura

[1] *Digital Negative (DNG) Specification.* Adobe Systems Incorporated, září 2012. Dostupné z: https://www.kronometric.org/phot/processing/DNG/dng_spec_1.4.0.0.pdf.

[2] *Camera Raw Formats.* Library of Congress, duben 2023. [online]. Dostupné z: https://www.loc.gov/preservation/digital/formats/fdd/fdd000241.shtml.

[3] Ammann, P. a Offutt, J. *Introduction to software testing.* 1. vyd. Cambridge University Press, 2008. ISBN 978-0-521-88038-1.

[4] astanin. GitHub, 2010. Zdrojový kód. Dostupné z: https://gist.github.com/astanin/626356.

[5] Karaimer, H. C. a Brown, M. S. *A Software Platform for Manipulating the Camera Imaging Pipeline.* European Converence on COmputer Vision, říjen 2016. [online]. Dostupné z: https://karaimer.github.io/camera-pipeline/paper/Karaimer_Brown_ECCV16.pdf.

[6] Karel Horák, T. Z. a. On Bayer demosaicing impact on image sharpness in lens quality assessment. *Proceedings II of the 28st Conference STUDENT EEICT 2022: Selected papers.* 2022, s. 291–296. DOI: 10.13164/eeict.2022.291. [online]. Dostupné z: http://hdl.handle.net/11012/208655.

[7] Khan, M. E. a Khan, F. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. *International Journal of Computer Applications.* The Science and Information Organization. 2012, sv. 3, č. 6, s. 1–4.

[8] Mathur, S. a Malik, S. Advancements in the V-Model. *International Journal of Computer Applications.* The Science and Information Organization. 2010, sv. 1, č. 12, s. 30–31.

[9] Mokhtar, R. a Khayyat, M. A Comparative Case Study of Waterfall and Agile Management. *SAR Journal.* Březen 2022, sv. 5. DOI: 10.18421/SAR51-07. [online]. Dostupné z: https://www.researchgate.net/publication/359538977.

[10] Myers, G. J., Badgett, T. a Sandler, C. *The art of software testing.* 3. vyd. John Wiley & Sons, Inc., 2012. ISBN 978-1-118-03196-4.

[11] Petersson, S., Grahn, H. a Rasmusson, J. Blind Correction of Lateral Chromatic Aberration in Raw Bayer Data. *IEEE Access.* 2021, sv. 9, s. 99455–99466. DOI: 10.1109/ACCESS.2021.3096201. [online]. Dostupné z: https://bth.diva-portal.org/smash/get/diva2:1584480/FULLTEXT01.pdf.

[12] Pihan, R. *Vše o formáru RAW, 1. díl.* Březen 2008. [online]. Dostupné z: https://www.digimanie.cz/vse-o-formatu-raw-1dil/2182.

[13] Pihan, R. *Vše o formáru RAW, 2. díl.* Březen 2008. [online]. Dostupné z: https://www.digimanie.cz/vse-o-formatu-raw-2dil/2191.

[14] Pihan, R. *Vše o formáru RAW, 3. díl.* Březen 2008. [online]. Dostupné z: https://www.digimanie.cz/vse-o-formatu-raw-3dil/2198.

[15] Rojas, R. *Global Computer Vision.* Svobodná univerzita v Berlíně. [online]. Dostupné z: https://www.inf.fu-berlin.de/lehre/WS02/robotik/Vorlesungen/Vorlesung2/ComputerVision-2.pdf.

[16] Vokolos, F. a Weyuker, E. Performance testing of software systems. *International Journal of Engineering and Technical Research.* 1998, s. 80–87. DOI: 10.1145/287318.287337. [online]. Dostupné z: https://dl.acm.org/doi/pdf/10.1145/287318.287337.

Příloha A

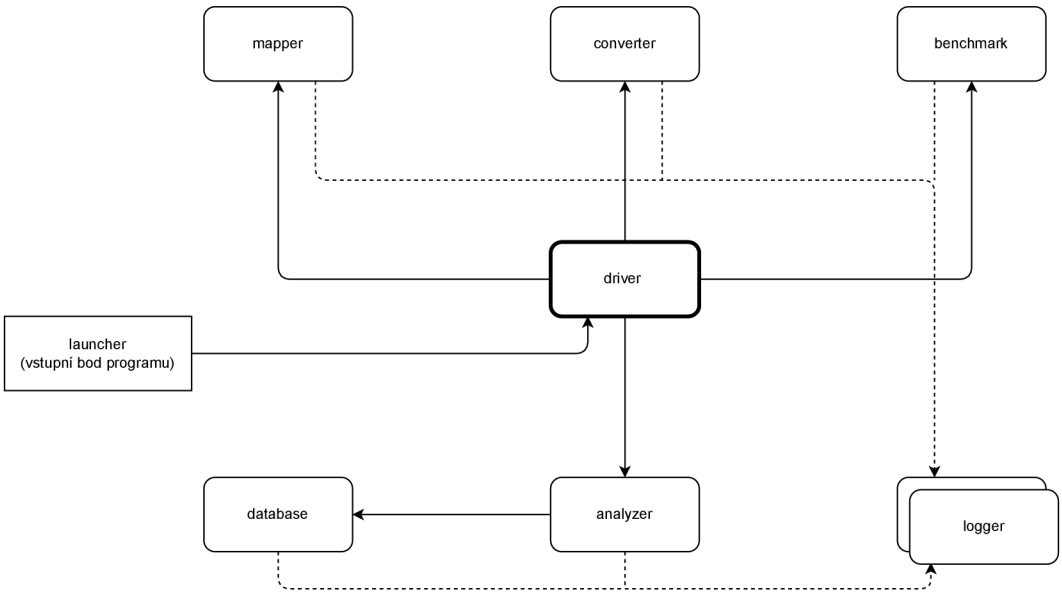
Existující typy RAW souborů, jejich koncovky a výrobci

Přípona souboru	Výrobce
.3fr	Hasselblad
.ari, .arw	Sony
.bay	Casio
.braw, .crw, .cr2, .cr3	Canon
.cap	Phase One
.data, .dcs, .dcr, .dng	Kodak
.drf	Epson
.eip, .erf	Epson
.fff	Hasselblad
.gpr	GoPro
.iiq	Phase One
.k25, .kdc	Kodak
.mdc, .mef, .mos, .mrw	Mamiya
.nef, .nrw	Nikon
.obm, .orf	Olympus
.pef, .ptx, .pxn	Pentax
.r3d, .raf, .raw, .rwl, .rw2, .rwz	Fujifilm
.sr2, .srf, .srw	Sony
.tif	Různé
.x3f	Sigma
.dng	Standardizace RAW formátu firmy Adobe

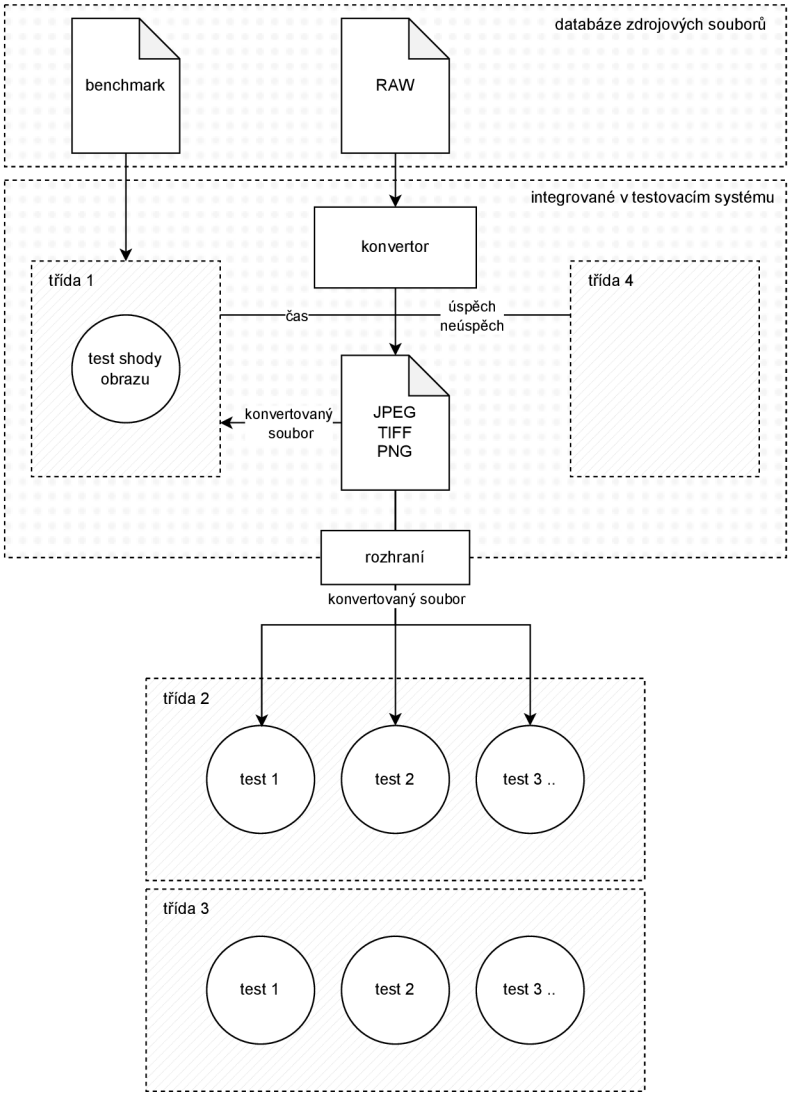
Tabulka A.1: Seznam všech používaných RAW formátů, jejich koncovky a výrobci[2]

Příloha B

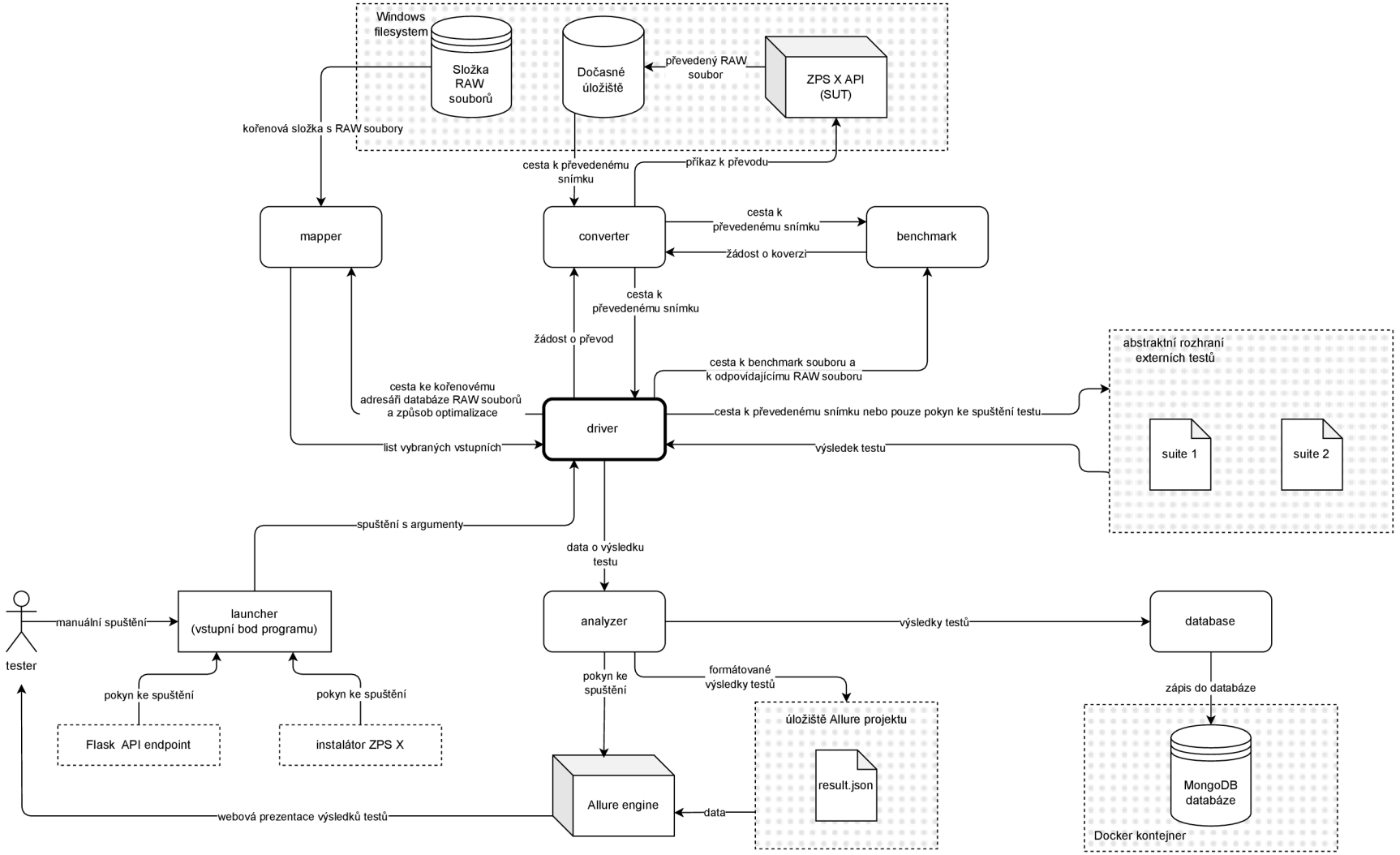
Dokumentační diagramy navrženého testovacího systému



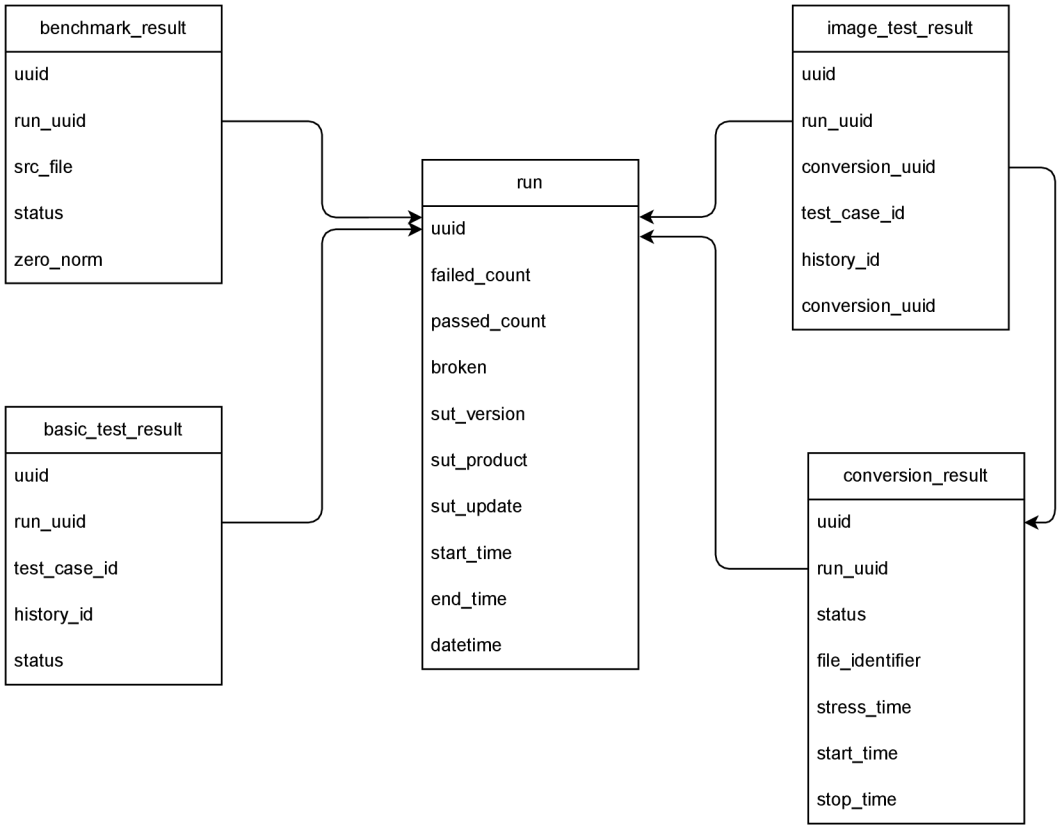
Obrázek B.1: Diagram sekvence inicializace jednotlivých objektů. Inicializátor si vždy drží referenci na inicializovaný objekt a pracuje s ním. Launcher v tomto diagramu není objekt, ale modul používáný pro spouštění testování.



Obrázek B.2: Struktura tříd testů v systému a generalizace jednoho převodu RAW souboru pro všechny testy. V případě třídy 4 se jedná pouze o vytvoření záznamu o stavu převodu.



Obrázek B.3: Strukturální diagram proudu dat systémem. Je vynechát modul `logger`, který je inicializován zvlášť pro každý funkcionální modul, aby byl diagram čitelnější.



Obrázek B.4: Diagram struktury NoSQL databáze a záznamy o jednotlivých datových aktérech v systému. Mapování záznamu na třídy testovacích případů (4.5): `basic_test_result` je 3. třída, `conversion_result` je 4. třída, `image_test_result` je druhá třída a `benchmark_result` je první třída testů v systému.

Příloha C

Informace o používání testovacího systému

C.1 Argumenty programu

```
    usage: launcher.py [-h] [-s SRC] [--suite SUITE] [-o OPTIMIZATION] [-m VENDOR]
                       [-c CAMERA] [-n] -v VERSION -b BINARIES_PATH -u UPDATE -P
                       PRODUCT [-l] [-S] [-p SUITES_PATH] [-t TEMP_PATH] [-r]
                       [-T TIME_STRESS] [-g] [-B]

ZTest - testing framework for Zoner Photo Studio X

options:
  -h, --help            show this help message and exit
  -s SRC, --src SRC     Path to the root folder of image database (Must be Win
                        filesystem) (default: default)
  --suite SUITE         Runs only given test suite given by the test suite
                        file. Default is all suites. (default: *)
  -o OPTIMIZATION, --optimization OPTIMIZATION
                        Sets input optimization level - must be one of
                        ['optimal', 'full'] (default: optimal)
  -m VENDOR, --vendor VENDOR
                        Sets tests just for a specific camera vendor. Uses
                        substring matching (default: all)
  -c CAMERA, --camera CAMERA
                        Sets test just for a specific camera. Uses substring
                        matching (default: all)
  -n, --no-report       Doesn't generate regression report if specified
                        (default: False)
  -v VERSION, --version VERSION
                        Specifies the SUT version (default: None)
  -b BINARIES_PATH, --binaries_path BINARIES_PATH
                        Specifies the path to SUT executables (default: None)
  -u UPDATE, --update UPDATE
                        Specifies the SUT release type from ['HOTFIX',
```

```
                      'FEATURE_UPDATE'] (default: None)
-P PRODUCT, --product PRODUCT
                      Specifies the SUT release channel (default: None)
-l, --log_to_file     Sets logger output to a file instead of stdout
                      (default: False)
-S, --serve           Serves a HTTP server that will trigger test run upon
                      successful request. If this option is specified,
                      nothing else happens. (default: False)
-p SUITES_PATH, --suites_path SUITES_PATH
                      Specifies a path to the folder containing individual
                      test suites in the form of python modules. (default:
                      default)
-t TEMP_PATH, --temp_path TEMP_PATH
                      Specifies a path to the temporary file storage.
                      Default is specified in config. (default: default)
-r, --report          Generates a Allure testing report when toggled.
                      (default: False)
-T TIME_STRESS, --time_stress TIME_STRESS
                      Specifies the time of cpu stress during ZRAW
                      conversion. Max is 6 seconds. (default: 0)
-g, --benchmark_generate
                      When specified, the benchmarks will be created for
                      each file in the SRC directory. (default: False)
-B, --benchmark_run   When specified, the system will run file compilations
                      tests against the benchmarking files. Make sure that
                      you have the benchmark files generated via -B switch.
                      (default: False)
```

C.2 Seznam fatálních chybových hlášek a jejich význam

`DB_UNINITIATED` - nelze načíst nastavení databáze ze souboru src/config.yml

`DB_CONNECTION_FAILED` - nelze se připojit k MongoDB databázi

`ANALYZER_INVALID_ALLURE_BIN_PATH` - systém Allure není spustitelný z cesty uvedené v config.yml

`ARG_INVALID_STRESS_TIME` - Nevalidní hodnota argumentu `-T`

`CONVERTER_TEMP_FOLDER_UNAVAILABLE` - složka pro ukládání dočasných souborů není dostupná

`CONVERTER_BINARIES_UNAVAILABLE` - binární soubory nebyly nalezeny v cestě zadané argumentem `-b`

`DRIVER_SETTING_DICT_ERROR` - obecná chyba pro nevalidní argumenty

`DRIVER_INVALID_YAML` - nevalidní struktura src/config.yml souboru

`BENCHMARK_DB_INCONSISTENT` - nekonzistentí benchmark soubory. Opravitelné generováním nových benchmark souborů.

`SRC_DB_UNAVAILABLE` - nevalidní cesta k databance RAW souborů zadaná přepínačem `-s`

`CAMERA_PRESET_UNAVAILABLE` - hodnota argumentu `-c` není validní název modelu fotoaparátu v databance RAW souborů.

`VENDOR_PRESET_UNAVAILABLE` - hodnota argumentu `-v` není validní název výrobce v databance RAW souborů

`INVALID_PRESET_TYPE` - nevalidní specifikace filtrování fotoaparátu nebo výrobce

`NOTHING_TO_FEED` - pod specifikovaným nastavením nebyly nalezeny žádné RAW soubory

`LOAD_ITERATOR_INVALID_PRESET` - nevalidní hodnota argumentu `-o`. Musí být full nebo optimal

C.3 Příklad obsahu konfiguračního souboru `config.yml`

```
config.yml
-------------------------------------------------------------------------
allure-path: "C:\\Users\\user\\ZTest\\allure\\"
# toto bývá výchozí instalace Allure při použití
# instalačních nástroje scoop
allure-bin: "C:\\Users\\user\\scoop\\shims\\allure.cmd"


# nastavení připojení k databázi
connection-string: "mongodb://localhost:27017/"
db-name: "ZTest"


# výchozí hodnoty pro driver objekt
suites-path: "C:\\Users\\user\\ZTest\\src\\suites\\"
temp-path: "C:\\Users\\user\\path\\to\\temp\\"

# nastavení optimalizace mapper, tedy vybírání souborů ze
# složek se vstupy podle indexu.
mapper-optimization:
    # každý název složky má svůj list indexů. Pokud jakýkoliv
    # index přeteče nebo podteče, přidají se pro jistotu
    # všechny soubory ze složky.

    # jen první soubor
    auto:
        - 0
    # soubory na indexu 0 až 8
    targets:
        - 0
        - 1
        - 2
        - 3
        - 4
        - 5
        - 6
        - 7
        - 8
    # první a poslední ISO soubor
    iso:
        - 0
        - -1
    # ze všech ostatní se berou všechny soubory
    other:
        - "all"
-------------------------------------------------------------------------
```

Hodnoty indexů v sekci `mapper-optimization` jsou popsány v sekci 4.7.3

C.4 Potřebné balíčky pro interpret Python

`requirements.txt`

```
--------------------------------------------------------------------------
PyYAML~=6.0
Pillow~=9.4.0
pymongo~=4.3.3
Flask~=2.2.3
waitress~=2.1.2
imageio~=2.25.0
scipy~=1.10.0
--------------------------------------------------------------------------
```

Balíčky, které je potřeba doinstalovat do standardního instalačního umístěný Python interpretu, popřípadě do virtuálního prostředí.

Příloha D

Hlášení výsledků pomocí rámce Allure

D.1 Příklad souboru prostředí testů

environment.properties

```
------------------------------------------------------------------------
version=19.2302.2.450
update=HOTFIX
product=STABLE_TEST
camera=all
vendor=all
optimization=optimal
------------------------------------------------------------------------
```

Soubor `environmnet.properties` obsahuje proměnné specifické pro testovanou verzi Zoner Photo Studia X a nastavení testovacího systému.

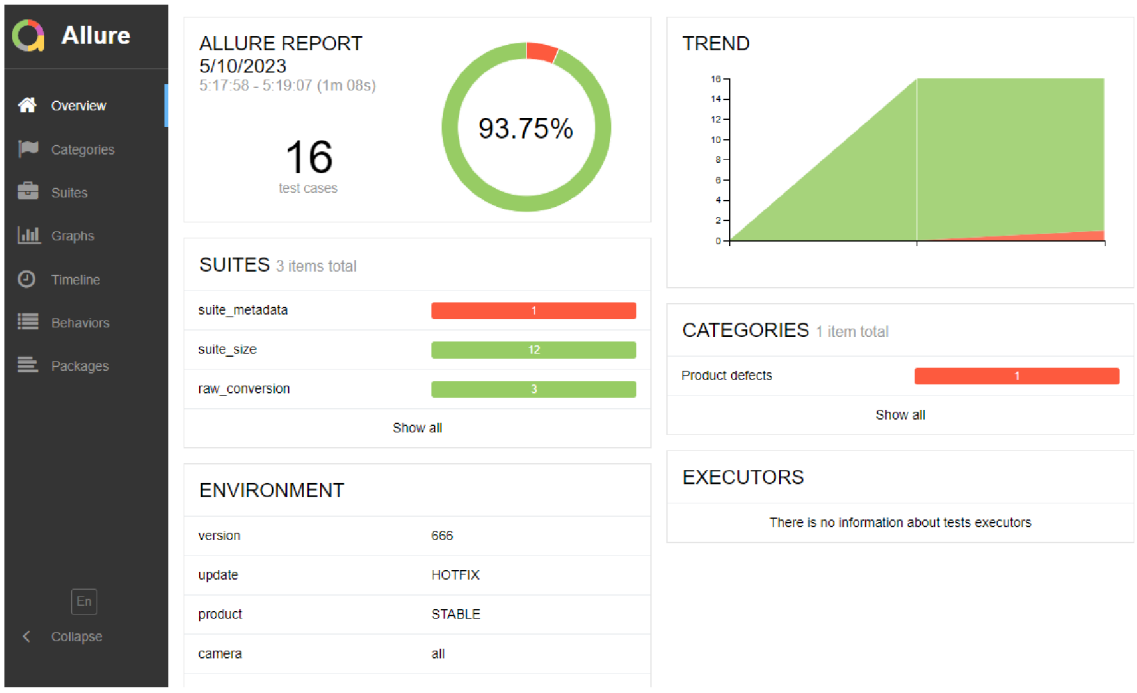
D.2 Příklad JSON souboru použitého pro export výsledku testu

f95f52c5-888b-4af1-ba9a-3ff86bbeb4ec.json

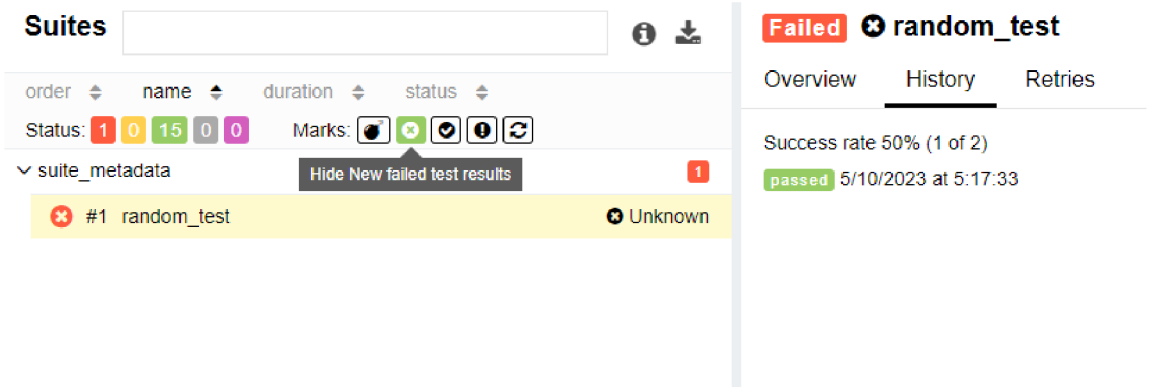
```
{
"name": "C:<cesta>\\Nikon\\Nikon Z fc\\auto\\DSC_0305.NEF",
"uuid": "f95f52c5-888b-4af1-ba9a-3ff86bbeb4ec",
"status": "passed",
"historyId": "raw_conversion#C:<cesta>\\Nikon\\Nikon Z fc\\auto\\DSC_0305.NEF",
"testCaseId": "raw_conversion#C:<cesta>\\Nikon\\Nikon Z fc\\auto\\DSC_0305.NEF",
"labels": [
    {
        "name": "suite",
        "value": "raw_conversion"
    },
    {
        "name": "host",
        "value": "DESKTOP-46CDR9M"
    },
    {
        "name": "thread",
        "value": "MainThread"
    },
    {
        "name": "framework",
        "value": "ZTest"
    }
],
"start": 1683693134671.5083,
"stop": 1683693144783.225
}
```

Tato struktura se používá při generování webové prezentace rámcem Allure.

D.3 Příklady webové prezentace výsledků



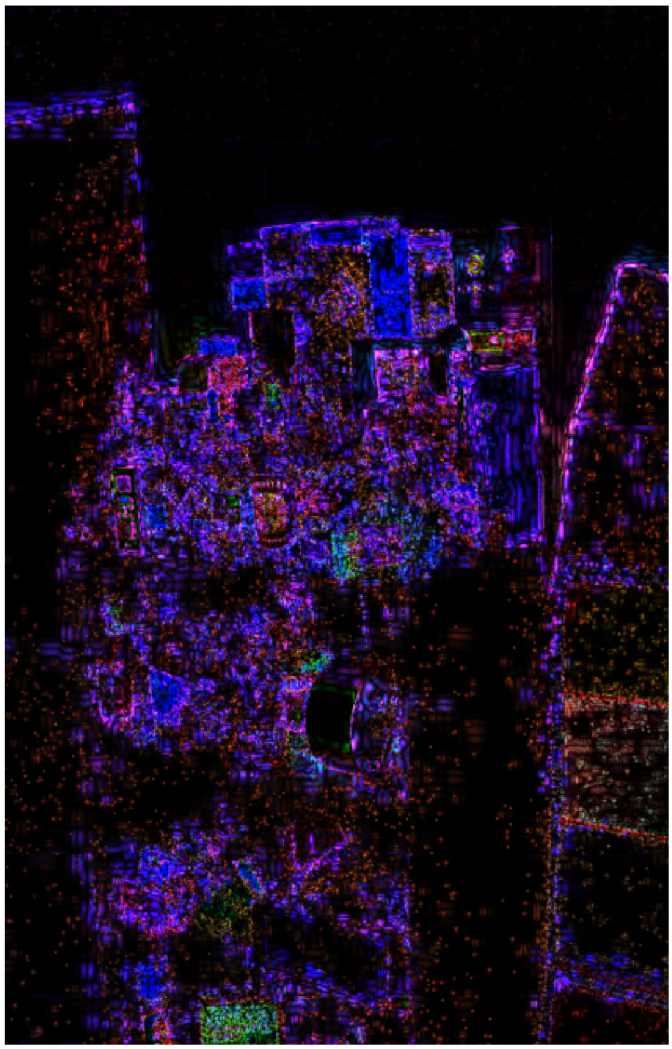
Obrázek D.1: Příklad reportu vygenerovaném po zkušebním běhu testů



Obrázek D.2: Demonstrace historie úspěšnosti jednotlivých testovacích případů. Na obrázku lze vidět různé možnosti filtrování. V této demnostraci bylo použito pouze několik testů, ale jakmile bude systém plně nasazen, lze očekávat velké množství výsledků a tato možnost vizualizace přijde vhod.

Příloha E

Chyby při konverzi



Obrázek E.1: Zvýraznění rozdílů na dvojici obrázů přeložených přes sebe. Zvýraznění bylo doděláno uměle, jelikož lidským okem nešel poznat rozdíl.